MULTI-LEVEL GRAPH MATCHING NETWORKS FOR DEEP AND ROBUST GRAPH SIMILARITY LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

While the celebrated graph neural networks yield effective representations for individual nodes of a graph, there has been relatively less success in extending to graph similarity learning. Recent works have considered either global-level graph-graph interactions or low-level node-node interactions, ignoring the rich cross-level interactions (*e.g.*, between nodes of a graph and the other whole graph). In this paper, we propose a Multi-level Graph Matching Network (MGMN) for computing the graph similarity between any pair of graph-structured objects in an end-to-end fashion. The proposed MGMN model consists of a node-graph matching network for effectively learning *cross-level interactions* between nodes of a graph and the other whole graph, and a siamese graph neural network to learn *global-level interactions* between two graphs. Furthermore, to bridge the gap of the lack of standard graph similarity learning benchmark, we have created and collected a set of datasets for both graph-graph classification and regression tasks with different sizes in order to evaluate the robustness of models. Our comprehensive experiments demonstrate that MGMN consistently outperforms state-of-the-art baselines on these graph similarity learning benchmarks.

1 INTRODUCTION

Learning a general similarity metric between arbitrary pairs of graph-structured objects is one of the key challenges in machine learning. Conceptually, graph matching can be broadly categorized into *exact* and *error-tolerant* graph matching techniques (Riesen et al., 2010; Dwivedi & Singh, 2018). For *exact* graph matching, a strict one-to-one correspondence is required between nodes and edges of two graphs, whereas *error-tolerant* graph matching techniques try to compute a similarity between two graphs. In some real-world applications, the constraint of *exact* graph matching is too rigid (*i.e.*, no need for strict correspondences), and thus we focus on the *error-tolerant* graph matching – the graph similarity problem to learn a similarity score between a pair of graph inputs.

Although graph neural networks (GNNs) have recently demonstrated to be a powerful class of neural networks for learning node embeddings of graphs on tasks ranging from node classifications, graph classifications to graph generations (Kipf & Welling, 2017; Ying et al., 2018; You et al., 2018; Chen et al., 2020), there is relatively less study on learning graph similarity using GNNs. A simple yet straightforward way is to use GNN to encode each graph as a vector and combine the vectors of two graphs to make a decision. This simple approach can be effective as the graph-level vector contains important information of a pair of graphs, but one obvious limitation is that this approach ignores finer-grained interactions among multi-level embeddings of two graphs.

Very recently, a few attempts have been made to take into account low-level interactions either by considering the histogram information or spatial patterns (with CNNs) of the node-wise similarity matrix of node embeddings (Bai et al., 2019; 2020), or by improving the node embeddings of one graph by incorporating the implicit attentive neighbors of another graph (Li et al., 2019). However, there are two significant challenges making these graph matching networks potentially ineffective: i) how to effectively learn richer cross-level interactions between nodes of a graph and the other whole graph; ii) how to integrate multi-level granularity (cross-level and global-level) of interactions between a pair of graphs for computing graph similarity in an end-to-end fashion;

Inspired by these observations, in this paper, we propose a **M**ulti-level **G**raph **M**atching **N**etwork (MGMN) for computing the graph similarity between any pair of graph-structured objects in an end-

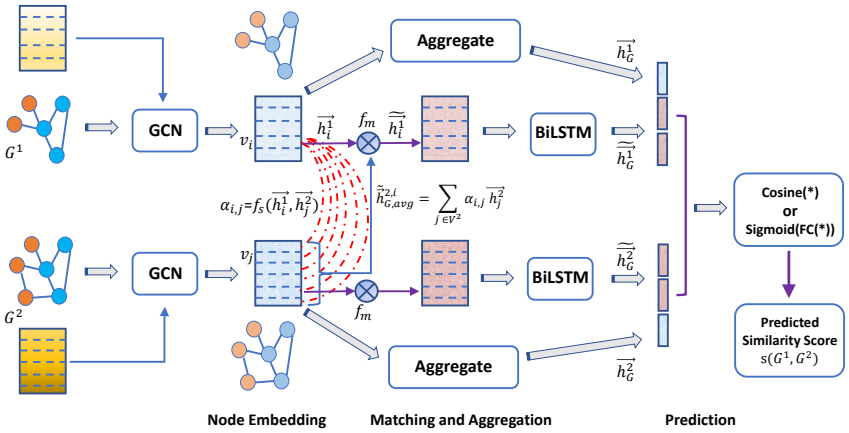

Figure 1: Overall architecture of the full model MGMN, consisting of two components: SGNN and NGMN. At the final prediction layer, we concatenate a total of 6 aggregated graph-level embedding vectors where the middle 4 (pink) are from NGMN and the other 2 (blue) are from SGNN.

to-end fashion. MGMN consists of a novel node-graph matching network (NGMN) for effectively learning *cross-level interaction features* between nodes of a graph and the other whole graph, and a siamese graph neural network (SGNN) for learning *global-level interaction features* between two graphs. Our final small prediction network leverage these feature vectors that are learned from both cross-level and global-level interactions to perform either graph-graph classification tasks or graph-graph regression tasks, respectively. Fig. 1 shows overall architecture of the proposed MGMN.

Furthermore, to bridge the gap of the lack of standard graph similarity learning benchmark, we have created and collected a set of datasets for both graph-graph classification and regression tasks with different sizes in order to evaluate the robustness of models. Our open graph similarity learning benchmark is partially inspired by the Open Graph Benchmark (Hu et al., 2020) and partially motivated by that existing works only consider either graph-graph classification task [1] (Li et al., 2019), or graph-graph regression task (Bai et al., 2019; 2020). One important aspect is previous work does not consider the impact of the size of input graphs, which often plays an important role in determining the performance of graph similarity learning. Motivated by this observation, we consider three different ranges of graph sizes to evaluate the robustness of models. To demonstrate the effectiveness of our model, we systematically evaluate MGMN on these graph similarity benchmarks for both the graph-graph classification and regression tasks. Our code and data are available for research purposes at `https://github.com/tinker467/mgmn`. In brief, we highlight our main contributions as follows:

- We propose a multi-level graph matching network (MGMN) for computing deep and robust graph similarity between any pair of graph-structured objects in an end-to-end fashion. In particular, our MGMN takes into account both cross-level and graph-level interactions between a pair of graphs.
- We present a novel node-graph matching network (NGMN) for effectively capturing the cross-level interactions between a node embedding of a graph and a corresponding attentive graph-level embedding of the other graph.
- Comprehensive experiments demonstrate that MGMN consistently outperforms state-of-the-art baselines for different tasks (*i.e.*, classification and regression) and also exhibits stronger robustness as the sizes of the two input graphs increase.

2 Problem Formulation

In this section, we briefly introduce the problem formulation. Given a pair of graph inputs (G^1, G^2), the aim of graph similarity learning in this paper is to produce a similarity score $y = s(G^1, G^2) \in \mathcal{Y}$.

[1] Noted that, the graph-graph classification tasks here are different from the general graph classification tasks (Ying et al., 2018; Ma et al., 2019) that only assign each graph with a label. Our graph-graph classification tasks learn a binary label (*i.e.*, similar or dissimilar) for pairs of two graphs instead of one graph.

The graph $G^1 = (\mathcal{V}^1, \mathcal{E}^1)$ is represented as a set of N nodes $v_i \in \mathcal{V}^1$ with a feature matrix $X^1 \in \mathcal{R}^{N\times d}$, edges $(v_i, v_j) \in \mathcal{E}^1$ (binary or weighted) formulating an adjacency matrix $A^1 \in \mathcal{R}^{N\times N}$, and a degree matrix $D^1_{ii} = \sum_j A^1_{ij}$. Similarly, the graph $G^2 = (\mathcal{V}^2, \mathcal{E}^2)$ is represented as a set of M nodes $v_i \in \mathcal{V}^2$ with a feature matrix $X^2 \in \mathcal{R}^{M\times d}$, edges $(v_i, v_j) \in \mathcal{E}^2$ (binary or weighted) formulating an adjacency matrix $A^2 \in \mathcal{R}^{M\times M}$, and a degree matrix $D^2_{ii} = \sum_j A^2_{ij}$. Note that, when performing the graph-graph classification tasks y is the class label $y \in \{-1, 1\}$; when performing the graph-graph regression tasks y is the similarity score $y \in (0, 1]$. We train our model based on a set of training triplet of structured input pairs and scalar output score $(G^1_1, G^2_1, y_1), ..., (G^1_n, G^2_n, y_n) \in \mathcal{G} \times \mathcal{G} \times \mathcal{Y}$ drawn from some fixed but unknown probability distribution in real applications.

3 Multi-level Graph Matching Networks

In this section, we detail the proposed Multi-level Graph Matching Network (MGMN), which consists of both Node-Graph Matching Network (NGMN) and Siamese Graph Neural Network (SGNN). The overall model architecture of MGMN is shown in Figure 1.

3.1 NGMN for Cross-level Interaction Learning

Existing work has considered either global-level graph-graph interactions or low-level node-node interactions, ignoring the rich cross-level interactions between two graphs. Inspired by these observations, we propose a novel node-graph matching network (NGMN) to effectively learn the cross-level node-graph interaction features between nodes of one graph and the other whole graph.

Node Embedding Layer. We use t-layer graph convolution networks (GCN) with the siamese network architecture (Bromley et al., 1994) to generate node embeddings $H^l = \{\vec{h}^l_i\}^{\{N,M\}}_{i=1} \in \mathcal{R}^{\{N,M\}\times d'}$ of both graphs G^1 and G^2,

$$H^l = \sigma\Big(\bar{A}^l \dots \sigma\Big(\bar{A}^l\, \sigma\Big(\bar{A}^l X^l W^{(0)}\Big) W^{(1)}\Big) \dots W^{(t-1)}\Big),\ l = \{1, 2\} \tag{1}$$

where $l = \{1, 2\}$ in the superscript of H^l, $\bar{A}^l$, X^l indicates it belongs to graph G^1 or G^2, N and M denote the number of nodes for both graphs G^1 and G^2, σ is the activation function, $\bar{A}^l = (\widetilde{D}^l)^{-\frac{1}{2}} \widetilde{A}^l (\widetilde{D}^l)^{-\frac{1}{2}}$ is the normalized Laplacian matrix for $\widetilde{A}^l = A^l + I_{\{N,M\}}$ depending on G^1 or G^2, and $W^{(i)}, i = \{0, 1, \dots, t-1\}$ are hidden weighted matrices of the i-th GCN layer.

Node-Graph Matching Layer. This layer is the key part of NGMN, which can effectively learn the cross-level interactions between nodes of a graph and the other whole graph. To build more tight interactions between the two graphs for learning the graph-level embedding of each other, we first calculate the cross-graph attention coefficients between the node $v_i \in \mathcal{V}^1$ in G^1 and all other nodes $v_j \in \mathcal{V}^2$ in G^2. Similarly, we calculate the cross-graph attention coefficients between the node $v_j \in \mathcal{V}^2$ in G^2 and all other nodes $v_i \in \mathcal{V}^1$ in G^1. These two cross-graph attention coefficients can be computed with an attention function f_s independently,

$$\alpha_{i,j} = f_s(\vec{h}^1_i, \vec{h}^2_j),\ v_j \in \mathcal{V}^2 \quad \text{and} \quad \beta_{j,i} = f_s(\vec{h}^2_j, \vec{h}^1_i),\ v_i \in \mathcal{V}^1 \tag{2}$$

where f_s is the attention function for computing the similarity score. For simplicity, we use cosine function in our experiments but other similarity metrics can be adopted as well. Then, from the view of the node in one graph, we try to learn the attentive graph-level embeddings of another graph. Specifically, we compute the attentive graph-level embeddings $\widetilde{\vec{h}}^{2,i}_{G,avg}$ of G^2 from the view of the node $v_i \in \mathcal{V}^1$ in G^1 by weighted averaging all node embeddings of G^2 with attentions. Thus, we compute these two attentive graph-level embeddings as follow.

$$\widetilde{\vec{h}}^{2,i}_{G,avg} = \sum_{j\in\mathcal{V}^2} \alpha_{i,j} \vec{h}^2_j \quad \text{and} \quad \widetilde{\vec{h}}^{1,j}_{G,avg} = \sum_{i\in\mathcal{V}^1} \beta_{j,i} \vec{h}^1_i. \tag{3}$$

Next, we define a multi-perspective matching function f_m to compute the similarity feature vector by comparing two vectors as follows,

$$\widetilde{\vec{h}}(k) = f_m(\vec{x}_1, \vec{x}_2, \vec{w}_k) = f_m(\vec{x}_1 \odot \vec{w}_k, \vec{x}_2 \odot \vec{w}_k),\ k = 1, \dots, \widetilde{d} \tag{4}$$

where $\widetilde{\vec{h}} \in \mathcal{R}^{\widetilde{d}}$ is a $\widetilde{d}$-dimension similarity feature vector, $W_m = \{\vec{w}_k\}_{k=1}^{\widetilde{d}} \in \mathcal{R}^{d' \times \widetilde{d}}$ is a trainable weight matrix and each $\vec{w}_i$ represents a perspective with a total $\widetilde{d}$ number of perspectives. Notably, f_m could be any similarity function and we use the cosine similarity metric in our experiments. It is worth noting that the proposed f_m essentially shares a similar spirit with multi-head attention (Vaswani et al., 2017), with the difference that multi-head attention uses $\widetilde{d}$ number of weighted matrices instead of vectors.

Thus, we use f_m to compare the i-th or j-th node embedding of a graph with the corresponding attentive graph-level embedding to capture the cross-level node-graph interactions. The resulting similarity feature vectors $\widetilde{\vec{h}}_i^1$ or $\widetilde{\vec{h}}_i^2 \in \mathcal{R}^{\widetilde{d}}$ (w.r.t node v_i in either G^1 or G^2) can thus be computed by,

$$\widetilde{\vec{h}}_i^1 = f_m(\vec{h}_i^1, \widetilde{\vec{h}}_{G,avg}^{2,i}, W_m),\ v_i \in \mathcal{V}^1 \quad \text{and} \quad \widetilde{\vec{h}}_j^2 = f_m(\vec{h}_j^2, \widetilde{\vec{h}}_{G,avg}^{1,j}, W_m),\ v_j \in \mathcal{V}^2 \tag{5}$$

After performing node-graph matching over all nodes for both G^1 and G^2, the newly produced interaction feature matrices $\widetilde{H}^1 = \{\widetilde{\vec{h}}_i^1\}_{i=1}^N \in \mathcal{R}^{N \times \widetilde{d}}$ and $\widetilde{H}^2 = \{\widetilde{\vec{h}}_j^2\}_{j=1}^M \in \mathcal{R}^{M \times \widetilde{d}}$ are ready to be fed into the following aggregation layer.

Aggregation Layer. To aggregate interaction features from the node-graph matching layer, we employ BiLSTM (Hochreiter & Schmidhuber, 1997) to aggregate the unordered feature embeddings,

$$\widetilde{\vec{h}}_G^l = \text{BiLSTM}\Big(\{\widetilde{\vec{h}}_i^l\}_{i=1}^{\{N,M\}}\Big), \quad l = \{1, 2\}. \tag{6}$$

where $\widetilde{\vec{h}}_G^l \in \mathcal{R}^{2\widetilde{d}}$ is computed by concatenating the last hidden vectors of two directions and represents the aggregated graph-level embedding for each graph G^1 and G^2. Although other aggregators can also be used, our extensive experiments show that the BiLSTM aggregator achieved consistent better performance over other aggregators (see Appendix A.5 for details). Similar LSTM-type aggregators have also been employed in the previous work (Hamilton et al., 2017; Zhang et al., 2019).

Prediction Layer. After the aggregated graph-level embeddings $\widetilde{\vec{h}}_G^1$ and $\widetilde{\vec{h}}_G^2$ are obtained, we then use these two graph-level embeddings to compute the similarity score of $s(G^1, G^2)$. As it is common to employ cosine similarity in the classification tasks (Xu et al., 2017; Gu et al., 2018), we directly compute the cosine similarity of two graph-level embeddings,

$$\widetilde{y} = s(G^1, G^2) = \text{cosine}(\widetilde{\vec{h}}_G^1, \widetilde{\vec{h}}_G^2) \tag{7}$$

Differently, the results of the regression tasks are continuous and are set in a range of (0,1]. Thus, for the regression tasks, we first concatenate the two graph embeddings into $[\widetilde{\vec{h}}_G^1, \widetilde{\vec{h}}_G^2]$, employ standard fully connected layers to gradually project the dimension of resulting vector down to 1, and finally perform the sigmoid function to enforce the similarity score in the range of (0,1],

$$\widetilde{y} = s(G^1, G^2) = \text{sigmoid}\Big(\text{MLP}\Big([\widetilde{\vec{h}}_G^1, \widetilde{\vec{h}}_G^2]\Big)\Big) \tag{8}$$

For both tasks, we train the NGMN model with the mean square error loss function to compare the computed similarity score $\widetilde{y}$ with the ground-truth similarity score y, *i.e.*, $\mathcal{L} = \frac{1}{n}\sum_{i=1}^n (\widetilde{y} - y)^2$.

3.2 SGNN FOR GLOBAL-LEVEL INTERACTION LEARNING

The graph-level embeddings contain important information of a graph. Therefore, learning graph-level interactions between two graphs could be an important component for learning the graph similarity of two graphs, and we propose the Siamese Graph Neural Network (SGNN) for that. Although Bai et al. (2019) also learned the graph-level embedding interaction, our SGNN is different from it in three aspects. 1) We apply a siamese network to learn node embeddings rather than independent; 2) SGNN only employs a simple aggregator to learn graph-level embedding while Bai et al. (2019) use a context-aware attention method; 3) SGNN directly employs the cosine function to learn the graph-graph interactions, whereas Bai et al. (2019) use a more sophisticated Neural Tensor Network.

Node Embedding Layer. Similar as described in Section 3.1, we also employ GCN to generate node embeddings $H^1 = \{\vec{h}_i^1\}_{i=1}^N \in \mathcal{R}^{N \times d'}$ and $H^2 = \{\vec{h}_j^2\}_{j=1}^M \in \mathcal{R}^{M \times d'}$ for graphs G^1 and G^2.

Conceptually, the node embedding layer of SGNN could be chosen to be an independent or shared node embedding layer (*i.e.*, GCN) with NGMN. As shown in Figure 1, our SGNN shares the same node embedding layer with NGMN due to two reasons: i) it reduces the number of parameters by half, which helps mitigate possible overfitting; ii) it maintains the consistency of the resulting node embeddings for both NGMN and SGNN, potentially leading to more aligned cross-level and graph-level interaction features.

Graph-level Embedding Aggregation Layer. With the computed node embeddings H^l for each graph, we need to aggregate them to formulate a corresponding graph-level embedding $\vec{h}_G^l$,

$$\vec{h}_G^l = \text{Aggregation}\Big(\{\vec{h}_i^l\}_{i=1}^{\{N,M\}}\Big), \quad l = \{1, 2\}. \tag{9}$$

We employ different aggregators such as element-wise max/mean pooling (Max/Avg), element-wise max/mean pooling following a fully connected layer (FCMax/FCAvg), and the BiLSTM aggregator.

Prediction Layer. After the aggregated graph embeddings $\vec{h}_G^1$ and $\vec{h}_G^2$ are obtained, we then use these two embeddings to compute the similarity score of $s(G^1, G^2)$. Just like the prediction layer in NGMN, we use Equations (7) and (8) to predict the similarity score for both classification and regression tasks. We also use the same mean square error loss function to train the model.

3.3 Discussions on the Full Model – MGMN

The full model MGMN combines the advantages of both NGMN and SGNN to capture both the cross-level node-graph interaction features and global-level graph-graph interaction features between two graphs. For the final prediction layer of MGMN, we have a total of 6 aggregated graph embedding vectors where 4 are $\widetilde{\vec{h}}_G^1$ and $\widetilde{\vec{h}}_G^2$ from NGMN, and another 2 are $\vec{h}_G^1$ and $\vec{h}_G^2$ from SGNN.

Complexity. The computation complexity of SGNN is $O((|\mathcal{E}^1| + |\mathcal{E}^2|)dd')$, where the most dominant computation is the sparse matrix-matrix operations in Equation (1). Similarly, the computational complexity of NGMN is $O(NMd + (N+M)d' + (N+M)dd')$, where the most computationally extensive operations are in Equations (3), (4), and (5). Compared to recently proposed work (Bai et al., 2019; 2020; Li et al., 2019), their computational complexities are highly comparable.

4 Experiments

4.1 Datasets, Experimental Setup, and Baselines

Classification Datasets:[2] we evaluate our model on the task of detecting a similarity score (*i.e.*, $y \in \{-1, 1\}$) between two binary functions, which is the heart of many binary security problems (Xu et al., 2017; Ding et al., 2019). As we represent binary with the control flow graph (CFG), detecting the similarity between two binaries can be cast as learning a similarity $s(G^1, G^2)$ between two CFGs: G^1 and G^2. We prepare two datasets from two popular open-source softwares: **FFmpeg** and **OpenSSL**. Besides, existing work does not consider the impact of the graph size on the performance. However, we find the larger the graph size is, the worse the performance is. Therefore, it is important to evaluate the robustness of graph similarity networks in this setting. We thus further split each dataset into 3 sub-datasets ([3, 200], [20,200], [50,200]) according to the range of graph sizes.

Regression Datasets: we evaluate our model on learning the graph edit distance (GED) (Gao et al., 2010; Riesen, 2015), which measures the structural similarity between two graphs. Formally, GED is defined as the cost of the least sequence of edits that transform one graph into another, where an edit can be an insertion/deletion of a node/edge. In our experiments, we normalize GED as $Y = e^{-[\frac{GED(G^1,G^2)}{(N+M)/2}]} \in (0, 1]$, and evaluate models on two datasets **AIDS700** and **LINUX1000** from (Bai et al., 2019). Table 1 shows the statistic for all datasets. We follow the standard train/validation/test split as previous work (Bai et al., 2019; Li et al., 2019) with more details in Appendix A.1.

Implementation Details. We implement our models using PyTorch 1.1 (Paszke et al., 2017) and train them with Adam optimizer (Kingma & Ba, 2015). In fact, the number of GCN layers required

[2]Although there are many benchmarks for the general graph classification tasks, these cannot be directly used in our graph-graph classification tasks as we cannot treat two graphs with the same labels as "similar".

Table 1: Summary statistics of datasets for both graph-graph classification & regression tasks.

Tasks	Datasets	Sub-datasets	# of Graphs	# of Functions	AVG # of Nodes	AVG # of Edges	Init Feature Dimensions
Classification	**FFmpeg**	[3, 200]	83,008	10,376	18.83	27.02	6
		[20, 200]	31,696	7,668	51.02	75.88	
		[50, 200]	10,824	3,178	90.93	136.83	
	OpenSSL	[3, 200]	73,953	4,249	15.73	21.97	6
		[20, 200]	15,800	1,073	44.89	67.15	
		[50, 200]	4,308	338	83.68	127.75	
Regression	**AIDS700**	-	700	-	8.90	8.80	29
	LINUX1000	-	1000	-	7.58	6.94	1

usually depends on real applications. To isolate the effect of over-tuning on different datasets and tasks, we choose the 3-layer GCN after examining how the number of GCN layers affect the performance (see Appendix A.4 for details). The output dimension of each GCN and $\widetilde{d}$ are both set to 100. For classification tasks, we train the model by running 100 epochs with 0.5e-3 learning rate. At each epoch, we build the pairwise training data as follows. For each graph G in the training subset, we obtain one positive pair $\{(G, G^{pos}), +1\}$ and a corresponding negative pair $\{(G, G^{neg}), -1\}$, where G^{pos} is randomly selected from all control flow graphs that compiled from the same source function as G, and G^{neg} is selected from other graphs. By default, each batch includes 5 positive and 5 negative pairs. For regression tasks, we train the model by running 10,000 iterations with a batch of 128 graph pairs with 5e-3 learning rate. Each pair is a tuple of $\{(G^1, G^2), s\}$, where s is the ground-truth GED between G^1 and G^2. Other implementation details can be found in Appendix A.2.1.

Baselines[3]: i) *SimGNN* (Bai et al., 2019) adopts GCN and applies 2 strategies to model the similarity between two graphs: one based on interactions between two graph-level embeddings, another based on histogram features from two sets of node embeddings; ii) *GMN* (Li et al., 2019) employs a variant of message passing neural networks and improves the node embeddings of one graph via incorporating the information of attentive neighborhoods of another graph; iii) *GraphSim* (Bai et al., 2020) extends SimGNN by turning the two sets of node embeddings into a similarity matrix and then processing the matrix with CNNs. Detailed experimental settings are given in Appendix A.2.2.

Note that we have two variants of the full model MGMN: **MGMN** (FCMax) and **MGMN** (BiLSTM), whose SGNN uses either the FCMax or BiLSTM aggregator, respectively. We repeat all experiments 5 times and report the mean and standard deviation of results, with the best in **bold**.

4.2 Comparison with Baseline Methods

Comparison of Graph-Graph Classification Tasks. For graph-graph classification tasks, we measure *AUC* (Bradley, 1997) of different models. As shown in Table 2, our models (both full model MGMN or the key component NGMN) clearly achieve state-of-the-art performance on all 6 sub-datasets for both **FFmpeg** and **OpenSSL**. Particularly when the graph size increases, both MGMN and NGMN show better and more *robust* performance than state-of-the-art baselines. In addition, compared with SGNN (Max), NGMN shows superior performance by a large margin, demonstrating the benefits of the proposed node-graph matching operation that captures the cross-level interaction features between node embeddings of a graph and the graph-level embedding of the other graph (see SGNN with other aggregators in Appendix A.3). MGMN (*i.e.*, NGMN+SGNN) further improves the performance of NGMN together with global-level interaction features learned from SGNN.

Comparison of Graph-Graph Regression Tasks. For graph-graph regression tasks, we evaluate the models using Mean Square Error (mse), Spearman's Rank Correlation Coefficient (ρ) (Spearman, 1904), Kendall's Rank Correlation Coefficient (τ) (Kendall, 1938), and precision at k (p@k). All results of both datasets are summarized in Table 3. Although GraphSim shows better performance than the other two baselines, our models (MGMN and its key component NGMN) outperform all baselines on both datasets in terms of most evaluation metrics. Moreover, compared with SGNN (Max), NGMN achieves much better performance (see more in Appendix A.3). It also highlights the importance of our proposed node-graph matching network, which could effectively

[3] As the three baselines only consider either graph-graph classification or regression tasks, we slightly adjust the last layer of the model or loss function of each baseline to make fair comparisons on both tasks.

Table 2: Summary of classification results in terms of AUC scores (%).

Model	FFmpeg			OpenSSL		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
SimGNN	95.38±0.76	94.31±1.01	93.45±0.54	95.96±0.31	93.58±0.82	94.25±0.85
GMN	94.15±0.62	95.92±1.38	94.76±0.45	96.43±0.61	93.03±3.81	93.91±1.65
GraphSim	97.46±0.30	96.49±0.28	94.48±0.73	96.84±0.54	94.97±0.98	93.66±1.84
SGNN	93.92±0.07	93.82±0.28	85.15±1.39	91.07±0.10	88.94±0.47	82.10±0.51
NGMN	97.73±0.11	98.29±0.21	96.81±0.96	96.56±0.12	**97.60±0.29**	92.89±1.31
MGMN (FCMax)	**98.07±0.06**	**98.29±0.10**	**97.83±0.11**	96.87±0.24	97.59±0.24	95.58±1.13
MGMN (BiLSTM)	97.56±0.38	98.12±0.04	97.16±0.53	**96.90±0.10**	97.31±1.07	**95.87±0.88**

capture cross-level node-graph interactions. MGMN (*i.e.*, SGNN+NGMN) further improves the performance of NGMN together with global-level interaction features learned from SGNN.

Table 3: Summary of regression results on **AIDS700** and **LINUX1000**.

Datasets	Model	mse (10^{-3})	ρ	τ	p@10	p@20
AIDS700	SimGNN	1.376±0.066	0.824±0.009	0.665±0.011	0.400±0.023	0.489±0.024
	GMN	4.610±0.365	0.672±0.036	0.497±0.032	0.200±0.018	0.263±0.018
	GraphSim	1.919±0.060	0.849±0.008	0.693±0.010	0.446±0.027	0.525±0.021
	SGNN	2.822±0.149	0.765±0.005	0.588±0.004	0.289±0.016	0.373±0.012
	NGMN	1.191±0.048	0.904±0.003	0.749±0.005	**0.465±0.011**	0.538±0.007
	MGMN (FCMax)	1.205±0.039	0.904±0.002	0.749±0.003	0.457±0.014	0.532±0.016
	MGMN (BiLSTM)	**1.169±0.036**	**0.905±0.002**	**0.751±0.003**	0.456±0.019	**0.539±0.018**
LINUX 1000	SimGNN	2.479±1.038	0.912±0.031	0.791±0.046	0.635±0.328	0.650±0.283
	GMN	2.571±0.519	0.906±0.023	0.763±0.035	0.888±0.036	0.856±0.040
	GraphSim	0.471±0.043	0.976±0.001	**0.931±0.003**	**0.956±0.006**	0.942±0.007
	SGNN	11.832±0.698	0.566±0.022	0.404±0.017	0.226±0.106	0.492±0.190
	NGMN	1.561±0.020	0.945±0.002	0.814±0.003	0.743±0.085	0.741±0.086
	MGMN (FCMax)	1.575±0.627	0.946±0.019	0.817±0.034	0.807±0.117	0.784±0.108
	MGMN (BiLSTM)	**0.439±0.143**	**0.985±0.005**	0.919±0.016	0.955±0.011	**0.943±0.014**

4.3 Ablation Studies

Different Attention Functions. As discussed in Section 3.1, the proposed multi-perspective matching function shares similar spirits with the multi-head attention mechanism (Vaswani et al., 2017), which makes it interesting to compare them. Therefore, we investigate the impact of these two different mechanisms for NGMN with classification results showed in Table 4. In our evaluation, the number of heads K is set to 6 because of the substantial consumption of resources of multi-head attention. Interestingly, our proposed multi-perspective attention mechanism consistently outperforms the results of the multi-head attention by quite a large margin. We suspect that our proposed multi-perspective attention uses vectors attention weights which may reduce the potential overfitting.

Table 4: Classification results of multi-perspectives versus multi-heads in terms of AUC scores(%).

Model	FFmpeg			OpenSSL		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
Multi-Perspectives ($\widetilde{d} = 100$)	**97.73±0.11**	**98.29±0.21**	**96.81±0.96**	**96.56±0.12**	**97.60±0.29**	**92.89±1.31**
Multi-Heads ($K = 6$)	91.18±5.91	77.49±5.21	68.15±6.97	92.81±5.21	85.43±5.76	56.87±7.53

Different Numbers of Perspectives. We further investigate the impact of different numbers of perspectives adopted by the node-graph matching layer of the NGMN model. Following the same settings of previous experiments, we only change the number of perspectives (*i.e.*, $\widetilde{d} = 50/75/100/125/150$) of NGMN. From Table 5, it is clearly seen that the AUC score of NGMN does not increase as the number of perspectives grows for classification tasks. Similar results of regression tasks can be found in Appendix A.6. We thus conclude that our model performance is not sensitive to the number of perspective $\widetilde{d}$ (from 50 to 150) and we make $\widetilde{d} = 100$ by default. **Different GNNs.** We investigate the impact of different GNNs including GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019), and GGNN (Li et al., 2016) adopted by the node embedding layer

Table 5: Classification results of different numbers of perspectives in terms of AUC scores(%).

Model	FFmpeg			OpenSSL		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
NGMN ($\widetilde{d} = 50$)	98.11±0.14	97.76±0.14	96.93±0.52	**97.38±0.11**	97.03±0.84	93.38±3.03
NGMN ($\widetilde{d} = 75$)	97.99±0.09	97.94±0.14	97.41±0.05	97.09±0.25	98.66±0.11	92.10±4.37
NGMN ($\widetilde{d} = 100$)	97.73±0.11	**98.29±0.21**	96.81±0.96	96.56±0.12	97.60±0.29	92.89±1.31
NGMN ($\widetilde{d} = 125$)	98.10±0.03	98.06±0.08	97.26±0.36	96.73±0.33	**98.67±0.11**	96.03±2.08
NGMN ($\widetilde{d} = 150$)	**98.32±0.05**	98.11±0.07	**97.92±0.09**	96.50±0.31	98.04±0.03	**97.13±0.36**

of NGMN for both classification and regression tasks. Table 6 presents the results of classification tasks (see the results of regression tasks in Appendix A.7). In general, the performance of different GNNs is quite similar for all datasets of both classification and regression tasks, which indicates that NGMN is not sensitive to the choice of GNNs in the node embedding layer. An interesting observation is that NGMN-GGNN performs even better than our default NGMN-GCN on both **FFmpeg** and **OpenSSL** datasets. This shows that our model can be further improved by adopting more advanced GNN models or choosing the most appropriate GNNs according to different application tasks.

Table 6: Classification results of different GNNs in terms of AUC scores (%).

Model	FFmpeg			OpenSSL		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
NGMN-GCN (Our)	97.73±0.11	98.29±0.21	96.81±0.96	96.56±0.12	97.60±0.29	92.89±1.31
NGMN-GraphSAGE	97.31±0.56	98.21±0.13	97.88±0.15	96.13±0.30	97.30±0.72	93.66±3.87
NGMN-GIN	97.97±0.08	98.06±0.22	94.66±4.01	96.98±0.20	97.42±0.48	92.29±2.23
NGMN-GGNN	**98.42±0.41**	**99.77±0.07**	**97.93±1.18**	**99.35±0.06**	**98.51±1.04**	**94.17±7.74**

5 Related Work

Conventional Graph Matching. As introduced in Section 1, graph matching can be categorized into *exact* and *error-tolerant* graph matching. In real-world applications, the constraint of exact graph matching is too rigid, and thus an amount of work has been proposed to solve the error-tolerant graph matching problem, which is usually quantified by a specific similarity metric, such as GED, maximum common subgraph (Bunke, 1997), or even more coarse binary similarity, according to different real applications. Particularly for GED, it is a well-studied NP-hard problem and suffers from exponential computational complexity and huge memory requirements for exact solutions in practice (McGregor, 1982; Zeng et al., 2009; Blumenthal & Gamper, 2018).

Graph Similarity Computation. Considering the great significance and challenge of computing the graph similarity, various approximation methods have been proposed for better accuracy and efficiency, including traditional heuristic methods (Gao et al., 2010; Riesen, 2015; Wu et al., 2019; Yoshida et al., 2019) and recent data-driven graph matching networks (Bai et al., 2019; 2020; Li et al., 2019), as detailed in the baselines of Section 4.1. Our work belongs to the graph matching networks, but differs from prior work in two main aspects: 1) unlike prior work only consider either graph-level or node-level interactions, our model successfully captures multi-level richer interactions between two graphs; 2) our work is the first one to systematically evaluate the performance on both graph-graph classification and regression tasks as well as the size of input graphs.

6 Conclusion and Future Work

In this paper, we presented a novel multi-level graph matching network (MGMN) for computing the graph similarity between any pair of graph-structured objects in an end-to-end fashion. In particular, we further proposed a new node-graph matching network for effectively learning cross-level interactions between two graphs beyond low-level node-node and global-level graph-graph interactions. Our extensive experimental results correlated the superior performance and robustness compared with state-of-the-art baselines on both graph-graph classification and regression tasks. One interesting future direction is to adapt our MGMN model for solving different real-world applications such as malware detection, text matching and entailment, and knowledge graph question answering.

REFERENCES

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pp. 384–392. ACM, 2019.

Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Thirty-Forth AAAI Conference on Artificial Intelligence*, 2020.

David B Blumenthal and Johann Gamper. On the exact computation of the graph edit distance. *Pattern Recognition Letters*, 2018.

Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 1997.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pp. 737–744, 1994.

Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.

Yu Chen, Lingfei Wu, and Mohammed J Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *NeurIPS*, 2020.

Steven HH Ding, Benjamin CM Fung, and Philippe Charland. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

Shri Prakash Dwivedi and Ravi Shankar Singh. Error-tolerant graph matching using node contraction. *Pattern Recognition Letters*, 116:58–64, 2018.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 933–944. IEEE, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

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. *arXiv preprint arXiv:2005.00687*, 2020.

Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 1938.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *International Conference on Learning Representations*, 2016.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. *ICML*, 2019.

Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 723–731, 2019.

James J McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12(1):23–34, 1982.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch, 2017.

Kaspar Riesen. Structural pattern recognition with graph edit distance. In *Advances in computer vision and pattern recognition*. Springer, 2015.

Kaspar Riesen, Xiaoyi Jiang, and Horst Bunke. Exact and inexact graph matching: Methodology and applications. In *Managing and Mining Graph Data*, pp. 217–247. Springer, 2010.

Kaspar Riesen, Sandro Emmenegger, and Horst Bunke. A novel software toolkit for graph edit distance computation. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pp. 142–151. Springer, 2013.

C Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 1904.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Xiaoli Wang, Xiaofeng Ding, Anthony KH Tung, Shanshan Ying, and Hai Jin. An efficient graph indexing method. In *2012 IEEE 28th International Conference on Data Engineering*, 2012.

Lingfei Wu, Ian En-Hsu Yen, Zhen Zhang, Kun Xu, Liang Zhao, Xi Peng, Yinglong Xia, and Charu Aggarwal. Scalable global alignment graph kernel using random features: From node embedding to graph embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1418–1428, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

Tomoki Yoshida, Ichiro Takeuchi, and Masayuki Karasuyama. Learning interpretable metric between graphs: Convex formulation and computation with graph mining. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1026–1036. ACM, 2019.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018.

Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

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*, 2019.

A APPENDIX

A.1 DATASETS

A.1.1 CLASSIFICATION DATASETS

In our evaluation, two binary functions that are compiled from the same source code but under different settings (architectures, compilers, optimization levels, etc) are considered to be semantically similar to each other. It is noted that one source code function, after compiled with different settings (architectures, compilers, optimization levels, etc), can generate various binary functions. To learn the similarity scores from pairs of binary functions, we represent those binary functions with control flow graphs, whose nodes represent the basic blocks (a basic block is a sequence of instructions without jumps) and edges represent control flow paths between these basic blocks. Thus, detecting the similarity between two binary functions can be cast as the problem of learning the similarity score $s(G^1, G^2)$ between two control flow graphs G^1 and G^2, where $s(G^1, G^2) = +1$ indicates G^1 and G^2 are similar; otherwise $s(G^1, G^2) = -1$ indicates dissimilar. We prepare two benchmark datasets generated from two popular open-source softwares: **FFmpeg** and **OpenSSL**, to evaluate our model on the graph-graph classification tasks.

For **FFmpeg**, we prepare the corresponding control flow graph (CFG) dataset as the benchmark dataset to detect binary function similarity. First, we compile *FFmpeg 4.1.4* using 2 different compilers *gcc 5.4.0* and *clang 3.8.0*, and 4 different compiler optimization levels (*O0-O3*), and generate 8 different binary files. Second, these 8 generated binaries are disassembled using IDA Pro,[4] which can produce CFGs for all disassembled functions. Finally, for each basic block in CFGs, we extract 6 block-level numeric features as the initial node representation based on IDAPython (a python-based plugin in IDA Pro).

OpenSSL is built from OpenSSL (v1.0.1f and v1.0.1u) using *gcc 5.4* in three different architectures (x86, MIPS, and ARM), and four different optimization levels (*O0-O3*). The **OpenSSL** dataset that we evaluate is previously released by Xu et al. (2017) and publicly available[5] with prepared 6 block-level numeric features.

Overall, for both **FFmpeg** and **OpenSSL** datasets, each node in the CFGs are initialized with 6 block-level numeric features: *# of string constants*, *# of numeric constants*, *# of total instructions*, *# of transfer instructions*, *# of call instructions*, and *# of arithmetic instructions*.

A.1.2 REGRESSION DATASETS

Instead of directly computing the graph edit distance (GED) between two graphs G^1 and G^2, we try to learn a similarity score $s(G^1, G^2)$, which is the normalized exponential of GED in the range of $(0, 1]$. To be specific, $s(G^1, G^2) = exp^{-normGED(G^1,G^2)}, normGED(G^1, G^2) = \frac{GED(G^1,G^2)}{(|G^1|+|G^2|)/2}$, where $|G^1|$ or $|G^2|$ denotes the number of nodes of G^1 or G^2, and $normGED(G^1, G^2)$ or $GED(G^1, G^2)$ denotes the normalized/un-normalized GED between G^1 and G^2.

We employ both **AIDS700** and **LINUX1000** released by Bai et al. (2019), which are publicly available.[6] Each dataset contains a set of graph pairs as well as their ground-truth GED scores, which are computed by exponential-time exact GED computation algorithm A^* (Hart et al., 1968; Riesen et al., 2013). As the ground-truth GEDs of another dataset **IMDB-MULTI** are provided with inexact approximations, we thus do not consider this dataset in our experiments.

AIDS700 is a subset of the *AIDS* dataset, a collection of AIDS antiviral screen chemical compounds from the Development Therapeutics Program (DTP) in the National Cancer Institute (NCI).[7] Originally, *AIDS* contains 42687 chemical compounds, where each of them can be represented as a graph with atoms as nodes and bonds as edges. To avoid calculating the ground-truth GED between two graphs with a large number of nodes, Bai et al. (2019) create the **AIDS700** dataset that contains 700

[4] IDA Pro disassembler, `https://www.hex-rays.com/products/ida/index.shtml`.

[5] `https://github.com/xiaojunxu/dnn-binary-code-similarity`.

[6] `https://github.com/yunshengb/SimGNN`.

[7] `https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data`

Table 7: Summary statistics of datasets for both classification & regression tasks.

Tasks	Datasets	Sub-datasets	# of Graphs	# of Functions	# of Nodes (Min/Max/AVG)	# of Edges (Min/Max/AVG)	# of Degrees (Min/Max/AVG)
classification	**FFmpeg**	[3, 200]	83,008	10,376	(3/200/18.83)	(2/332/27.02)	(1.25/4.33/2.59)
		[20, 200]	31,696	7,668	(20/200/51.02)	(20/352/75.88)	(1.90/4.33/2.94)
		[50, 200]	10,824	3,178	(50/200/90.93)	(52/352/136.83)	(2.00/4.33/3.00)
	OpenSSL	[3, 200]	73,953	4,249	(3/200/15.73)	(1/376/21.97)	(0.12/3.95/2.44)
		[20, 200]	15,800	1,073	(20/200/44.89)	(2/376/67.15)	(0.12/3.95/2.95)
		[50, 200]	4,308	338	(50/200/83.68)	(52/376/127.75)	(2.00/3.95/3.04)
regression	**AIDS700**	-	700	-	(2/10/8.90)	(1/14/8.80)	(1.00/2.80/1.96)
	LINUX1000	-	1000	-	(4/10/7.58)	(3/13/6.94)	(1.50/2.60/1.81)

graphs with 10 or fewer nodes. For each graph in **AIDS700**, every node is labeled with the element type of its atom and every edge is unlabeled (*i.e.*, bonds features are ignored).

LINUX1000 is also a subset dataset of Linux that was introduced in Wang et al. (2012). The original Linux dataset is a collection of 48747 program dependence graphs generated from Linux kernel. In this case, each graph is a static representation of data flow and control dependency within one function, with each node assigned to one statement and each edge describing the dependency between two statements. For the same reason as above that avoiding calculating the ground-truth GED between two graphs with a large number of nodes, the **LINUX1000** dataset used in Bai et al. (2019) is randomly selected and contains 1000 graphs with 10 or fewer nodes. For each graph in **LINUX1000**, both nodes and edges are unlabeled.

For both classification and regression datasets, Table 7 provides more detailed statistics. In our evaluation, for the classification tasks, we split each dataset into three disjoint subsets of *binary functions* for training/validation/testing. In the regression tasks, we first split graphs of each dataset into training, validation, and testing sets, and then build the pairwise training/validation/testing data as the previous work (Bai et al., 2019).

A.2 More Experimental Setup

A.2.1 Other experimental settings for our models

For SGNN, we use three GCN layers in the node embedding layer and each of the GCNs has an output dimension of 100. We use ReLU as the activation function along with a dropout layer after each GCN layer with the dropout rate being 0.1. In the graph-level embedding aggregation layer of SGNN, we can employ different aggregation functions (i,e., Max, FCMax, Avg, FCAvg, and BiLSTM) as stated previously in Section 3.2. For NGMN, We set the number of perspectives $\widetilde{d}$ to 100. We also employed different aggregation functions similar to SGNN and found that BiLSTM consistently performs better than others (see Appendix A.5). Thus, for NGMN, we take BiLSTM as the default aggregation function and we make its hidden size equal to the dimension of node embeddings. For each graph, we concatenate the last hidden vector of two directions of BiLSTM, which results in a 200-dimension vector as the graph embedding.

Noted that all experiments are conducted on a PC equipped with 8 Intel Xeon 2.2GHz CPU and one NVIDIA GTX 1080 Ti GPU.

A.2.2 Detailed experimental settings for baseline models

In principle, we follow the same experimental settings as the baseline methods of their original papers and adjust a few settings to fit specific tasks. For instance, **SimGNN** is originally used for graph-graph regression tasks, we modify the final layer of model architecture so that it can be used to evaluate graph-graph classification tasks fairly. Thus, detailed experimental settings of all three baseline methods for both classification and regression tasks are given as follows.

SimGNN: SimGNN firstly adopts three-layer GCN to encode each node of a pair of graphs into a vector. Then, SimGNN employs a two-stage strategy to model the similarity between the two

graphs: i) it uses Neural Tensor Network (NTN) module to interact two graph-level embeddings that are aggregated by a node attention mechanism; ii) it uses the histogram features extracted from the pairwise node-node similarity scores. Finally, the features learned from the two-stage strategy are concatenated to feed into multiple fully connected layers to obtain a final prediction.

For the graph-graph regression tasks, the output dimensions for the three-layer GCNs are 64, 32, and 16, respectively. The number of K in NTN and the number of histogram bins are both set to 16. Four fully connected layers are employed to reduce the dimension of concatenated results from 32 to 16, 16 to 8, 8 to 4, 4 to 1. As for training, the mean square error (MSE) loss function is used to train the model with Adam optimizer. The learning rate is set to 0.001 and the batch size is set to 128. We set the number of iterations to 10,000 and select the best model based on the lowest validation loss.

To fairly compare our model with SimGNN in evaluating graph-graph classification tasks, we adjust the settings of SimGNN as follows. We follow the same architecture of SimGNN in regression tasks except that the output dimension of the last connected layer is set to 2. We apply a softmax operation over the output of SimGNN to get the predicted binary label for the graph-graph classification tasks. As for training, we use the cross-entropy loss function to train our model and set the number of epochs to 100. Other training hyper-parameters are kept the same as the regression tasks.

GMN: The spirit of GMN is improving the node embeddings of one graph by incorporating the implicit neighbors of another graph through a soft attention mechanism. GMN follows a similar model architecture of the neural message passing network with three components: an encoder layer that maps the node and edge to initial vector features of node and edge, a propagation layer further update the node embeddings through proposed strategies, and an aggregator that compute a graph-level representation for each graph.

For the graph-graph classification tasks, we use 1-layer MLP as the node/edge encoder and set the number of rounds of propagation to 5. The dimension of the node feature is set to 32, and the dimension of graph-level representation is set to 128. The Hamming distance is employed to compute the distance of two graph-level representation vectors. Based on the Hamming distance, we train the model with the margin-based pairwise loss function for 100 epochs in which validation is carried out per epoch. Adam optimizer is used with the learning rate of 0.001 and batch size 10.

In order to enable fair comparisons with GMN for graph-graph regression tasks, we adjust the GMN architecture by concatenating the graph-level representation of two graphs and feeding it into a four-layer fully connected layers like SimGNN so that the final output dimension is reduced to 1. As for training, we use mean square loss function with batch size 128. Other settings remain the same as the classification tasks.

GraphSim: The main idea of GraphSim is to convert the graph similarity computation problems into pattern recognition problems. GraphSim first employs GCN to generate node embeddings of the pair of graphs, then turns the two sets of node embedding into a similarity matrix consisting of the pairwise node-node interaction similarity scores, feeds these matrics into convolutional neural networks (CNN), and finally concatenates the results of CNN to multiple fully connected layers to obtain a final predicted graph-graph similarity score.

For the graph-graph regression tasks, three layers of GCN are employed with each output dimension being set to 128, 64, and 32, respectively. The following architecture of CNNs is used: $Conv(6, 1, 1, 16)$, $maxpool(2)$, $Conv(6, 1, 16, 32)$, $maxpool(2)$, $Conv(5, 1, 32, 64)$, $maxpool(2)$, $Conv(5, 1, 64, 128)$, $maxpool(3)$, $Conv(5, 1, 128, 128)$, $maxpool(3)$. Numbers in $Conv()$ and $maxpool()$ indicates $Conv(window\_size, kernel\_stride, input\_channels, output\_channels)$ and $maxpool(pooling\_size)$. Eight fully connected layers are used to reduce the dimension of the concatenated results from CNNs, from 384 to 256, 256 to 128, 128 to 64, 64 to 32, 32 to 16, 16 to 8, 8 to 4, 4 to 1. As for training, the mean square error (MSE) loss function is used to train the model with Adam optimizer. The learning rate is set to 0.001 and the batch size is set to 128. Similar to SimGNN, we set the number of iterations to 10,000 and select the best model based on the lowest validation loss.

To make a fair comparison of our model with GraphSim in our evaluation, we also adjust GraphSim to solve the graph-graph classification tasks. We follow the same architecture of GraphSim in regression tasks except that seven connected layers are used instead of eight. The output dimension

of final connected layers is set to 2, and we apply a softmax operation over it to get the predicted binary label for the classification tasks. As for training, we use the cross-entropy loss function to train our model and set the number of epochs to 100. Other training hyper-parameters are kept the same as the regression tasks.

A.2.3 DETAILED EXPERIMENTAL SETUP FOR DIFFERENT GNNS

When performing experiments to see how different GNNs affect the performance of NGMN, we only replace GCN with GraphSAGE, GIN, and GGNN using the geometric deep learning library - PyTorch Geometric[8]. More specifically, for GraphSAGE, we used a 3-layer GraphSAGE GNN with their output dimensions all set to 100. For GIN, we used 3 GIN modules with a 1-layer MLP with output dimension 100 as the learnable function. For GGNN, we used 3 one-layer propagation models to replace the 3 GCNs in our original setting and also set their output dimensions to 100.

A.3 SGNN WITH DIFFERENT AGGREGATION FUNCTIONS FOR BOTH CLASSIFICATION & REGRESSION TASKS

To further compare our models with the SGNN models, we train and evaluate several SGNN models with different aggregation functions, such as Max, FCMax, Avg, FCAvg, and BiLSTM. The classification results and regression results are summarized in Table 8 and Table 9, respectively. For both classification and regression tasks, our models (the full model MGMN and key component NGMN) show statistically significant improvement over all SGNN models with different aggregation functions, which indicates the advantage of the proposed node-graph matching network.

Table 8: Classification results of SGNN models with different aggregation functions VS. NGMN and MGMN in terms of AUC scores (%).

Model	**FFmpeg**			**OpenSSL**		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
SGNN (BiLSTM)	96.92±0.13	97.62±0.13	96.35±0.33	95.24±0.06	96.30±0.27	93.99±0.62
SGNN (Max)	93.92±0.07	93.82±0.28	85.15±1.39	91.07±0.10	88.94±0.47	82.10±0.51
SGNN (FCMax)	95.37±0.04	96.29±0.14	95.98±0.32	92.64±0.15	93.79±0.17	93.21±0.82
SGNN (Avg)	95.61±0.05	96.09±0.05	96.70±0.13	92.89±0.09	93.90±0.24	94.12±0.35
SGNN (FCAvg)	95.18±0.03	95.74±0.15	96.43±0.16	92.70±0.09	93.72±0.19	93.49±0.30
NGMN	97.73±0.11	98.29±0.21	96.81±0.96	96.56±0.12	**97.60±0.29**	92.89±1.31
MGMN (Max)	97.44±0.32	97.84±0.40	97.22±0.36	94.77±1.80	97.44±0.26	94.06±1.60
MGMN (FCMax)	**98.07±0.06**	**98.29±0.10**	**97.83±0.11**	96.87±0.24	97.59±0.24	95.58±1.13
MGMN (BiLSTM)	97.56±0.38	98.12±0.04	97.16±0.53	**96.90±0.10**	97.31±1.07	**95.87±0.88**

A.4 NGMN WITH DIFFERENT NUMBER OF GNN LAYERS FOR BOTH CLASSIFICATION & REGRESSION TASKS

We also examine how the number of GNN (*i.e.*, GCN) layers would affect the performance of our models for both graph-graph classification and regression tasks. Follow the same default experimental settings (see Section 4.1 for details), we only change the number of GCN layers in the node embeddings layer of NGMN. Specifically, we change the number of layers form 1, 2, 3, to 4, and summarize the experimental results in Table 10 for graph-graph classification tasks as well as Table 11 for regression tasks.

It can be observed from Table 10 that the NGMN model with more GCN layers (*i.e.*, 3-layer and 4-layer) provides better and comparatively stable performance for all sub-datasets for both **FFmpeg** and **OpenSSL**, while NGMN with less GCN layers (*i.e.*, 1-layer or 2-layer) show inferior performance on some sub-datasets. For instance, NGMN with 1-layer performs extremely poorly on the [20, 200] and [50, 200] sub-datasets of both **FFmpeg** and **OpenSSL**, and NGMN with 2-layer runs poorly on the [20, 200] and [50, 200] sub-datasets of **FFmpeg** as well as [50, 200] sub-dataset of **OpenSSL**. From Table 11, NGMN with different number of layers exhibits similar results and none

[8] https://pytorch-geometric.readthedocs.io

Table 9: Regression results of SGNN models with different aggregation functions VS. NGMN and MGMN on **AIDS700** and **LINUX1000**.

Datasets	Model	mse (10^{-3})	ρ	τ	p@10	p@20
AIDS700	SGNN (BiLSTM)	1.422±0.044	0.881±0.005	0.718±0.006	0.376±0.020	0.472±0.014
	SGNN (Max)	2.822±0.149	0.765±0.005	0.588±0.004	0.289±0.016	0.373±0.012
	SGNN (FCMax)	3.114±0.114	0.735±0.009	0.554±0.008	0.278±0.021	0.364±0.017
	SGNN (Avg)	1.453±0.015	0.876±0.002	0.712±0.002	0.353±0.007	0.444±0.012
	SGNN (FCAvg)	1.658±0.067	0.857±0.007	0.689±0.008	0.305±0.018	0.399±0.021
	NGMN	1.191±0.048	0.904±0.003	0.749±0.005	**0.465±0.011**	0.538±0.007
	MGMN (Max)	1.210±0.020	0.900±0.002	0.743±0.003	0.461±0.012	0.534±0.009
	MGMN (FCMax)	1.205±0.039	0.904±0.002	0.749±0.003	0.457±0.014	0.532±0.016
	MGMN (BiLSTM)	**1.169±0.036**	**0.905±0.002**	**0.751±0.003**	0.456±0.019	**0.539±0.018**
LINUX 1000	SGNN (BiLSTM)	2.140±1.668	0.935±0.050	0.825±0.100	0.978±0.012	0.965±0.007
	SGNN (Max)	11.832±0.698	0.566±0.022	0.404±0.017	0.226±0.106	0.492±0.190
	SGNN (FCMax)	17.795±0.406	0.362±0.021	0.252±0.015	0.239±0.000	0.241±0.000
	SGNN (Avg)	2.343±0.453	0.933±0.012	0.790±0.017	0.778±0.048	0.811±0.050
	SGNN (FCAvg)	3.211±0.318	0.909±0.004	0.757±0.008	0.831±0.163	0.813±0.159
	NGMN	1.561±0.020	0.945±0.002	0.814±0.003	0.743±0.085	0.741±0.086
	MGMN (Max)	1.054±0.086	0.962±0.003	0.850±0.008	0.877±0.054	0.883±0.047
	MGMN (FCMax)	1.575±0.627	0.946±0.019	0.817±0.034	0.807±0.117	0.784±0.108
	MGMN (BiLSTM)	**0.439±0.143**	**0.985±0.005**	**0.919±0.016**	**0.955±0.011**	**0.943±0.014**

of them runs particularly better than the others on both **AIDS700** and **LINUX1000** datasets in terms of all evaluation metrics for graph-graph regression tasks.

These observations indicate that the number of GCN layers that are required in our models depends on the different graph datasets *i.e.*, different real applications. Thus, to avoid over-tuning this hyper-parameter (*i.e.*, number of GCN layers) on different datasets and different tasks as well as take the resource consumption of training models, we choose the three-layers GCN as the default for the node embedding layers in our models.

Table 10: Classification results of NGMN models with different numbers of GNN layers in terms of AUC scores (%).

Model	**FFmpeg**			**OpenSSL**		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
NGMN-(1 layers)	97.84±0.08	71.05±2.98	75.05±17.20	97.51±0.24	88.87±4.79	77.72±7.00
NGMN-(2 layers)	**98.03±0.15**	84.72±12.60	90.58±10.12	**97.65±0.10**	95.78±3.46	86.39±8.16
NGMN-(3 layers)	97.73±0.11	**98.29±0.21**	96.81±0.96	96.56±0.12	97.60±0.29	92.89±1.31
NGMN-(4 layers)	97.96±0.22	98.06±0.13	**97.94±0.15**	96.79±0.21	**98.21±0.31**	**93.40±1.78**

Table 11: Regression results of NGMN models with different numbers of GCN layers on **AIDS700** and **LINUX1000**.

Datasets	Model	mse (10^{-3})	ρ	τ	p@10	p@20
AIDS 700	NGMN-(1 layers)	1.297±0.025	0.895±0.001	0.737±0.002	0.414±0.011	0.498±0.006
	NGMN-(2 layers)	**1.127±0.015**	**0.908±0.001**	**0.755±0.002**	**0.479±0.009**	**0.555±0.006**
	NGMN-(3 layers)	1.191±0.048	0.904±0.003	0.749±0.005	0.465±0.011	0.538±0.007
	NGMN-(4 layers)	1.345±0.098	0.887±0.009	0.727±0.012	0.401±0.034	0.491±0.029
LINUX 1000	NGMN-(1 layers)	**1.449±0.234**	0.943±0.013	0.817±0.018	0.750±0.070	**0.786±0.065**
	NGMN-(2 layers)	1.525±0.119	**0.948±0.003**	**0.818±0.005**	0.706±0.076	0.736±0.039
	NGMN-(3 layers)	1.561±0.020	0.945±0.002	0.814±0.003	0.743±0.085	0.741±0.086
	NGMN-(4 layers)	1.677±0.248	0.943±0.008	0.810±0.013	**0.758±0.063**	0.765±0.071

A.5 NGMN WITH DIFFERENT AGGREGATION FUNCTIONS FOR BOTH CLASSIFICATION & REGRESSION TASKS

We investigate the impact of different aggregation functions adopted by the aggregation layer of NGMN model for both classification and regression tasks. Following the default and same settings of previous experiments, we only change the aggregation layer of NGMN and use five possible aggregation functions: Max, FCMax, Avg, FCAvg, LSTM, and BiLSTM. As can be observed from Table 12 and Table 13, BiLSTM offers superior performance on all datasets for both classification and regression tasks in terms of most evaluation metrics. Therefore, we take BiLSTM as the default aggregation function for NGMN, and fix it for the NGMN part in MGMN models.

Table 12: Classification results of NGMN models with different aggregation functions in terms of AUC scores (%).

Model	**FFmpeg**			**OpenSSL**		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
NGMN (Max)	73.74±8.30	73.85±1.76	77.72±2.07	67.14±2.70	63.31±3.29	63.02±2.77
NGMN (FCMax)	97.28±0.08	96.61±0.17	96.65±0.30	95.37±0.19	96.08±0.48	**95.90±0.73**
NGMN (Avg)	85.92±1.07	83.29±4.49	85.52±1.42	80.10±4.59	70.81±3.41	66.94±4.33
NGMN (FCAvg)	95.93±0.21	73.90±0.70	94.22±0.06	93.38±0.80	94.52±1.16	94.71±0.86
NGMN (LSTM)	97.16±0.42	97.02±0.99	84.65±6.73	96.30±0.69	97.51±0.82	89.41±8.40
NGMN (BiLSTM)	**97.73±0.11**	**98.29±0.21**	**96.81±0.96**	**96.56±0.12**	**97.60±0.29**	92.89±1.31

Table 13: Regression results of NGMN models with different aggregation functions on **AIDS700** and **LINUX1000**.

Datasets	Model	mse (10^{-3})	ρ	τ	p@10	p@20
AIDS 700	NGMN (Max)	2.378±0.244	0.813±0.015	0.642±0.013	**0.578±0.199**	**0.583±0.169**
	NGMN (FCMax)	2.220±1.547	0.808±0.145	0.656±0.122	0.425±0.078	0.504±0.064
	NGMN (Avg)	1.524±0.161	0.880±0.010	0.717±0.012	0.408±0.044	0.474±0.027
	NGMN (FCAvg)	1.281±0.075	0.895±0.006	0.737±0.008	0.453±0.015	0.527±0.016
	NGMN (LSTM)	1.290±0.037	0.895±0.004	0.737±0.005	0.448±0.007	0.520±0.012
	NGMN (BiLSTM)	**1.191±0.048**	**0.904±0.003**	**0.749±0.005**	0.465±0.011	0.538±0.007
LINUX 1000	NGMN (Max)*	16.921±0.000	-	-	-	-
	NGMN (FCMax)	4.793±0.262	0.829±0.006	0.665±0.011	**0.764±0.170**	**0.767±0.166**
	NGMN (Avg)	4.050±0.594	0.888±0.008	0.719±0.012	0.501±0.093	0.536±0.112
	NGMN (FCAvg)	6.953±0.195	0.897±0.004	0.736±0.005	0.499±0.126	0.509±0.129
	NGMN (LSTM)	1.535±0.096	0.945±0.004	0.813±0.007	0.695±0.064	0.698±0.081
	NGMN (BiLSTM)	**1.561±0.020**	**0.945±0.002**	**0.814±0.003**	0.743±0.085	0.741±0.086

* As all duplicated experiments running on this setting do not converge in their training processes, their corresponding result metrics cannot be calculated.

A.6 NGMN WITH DIFFERENT NUMBER OF PERSPECTIVES ON REGRESSION TASKS

As a supplement to Table 5 in Section 4.3, Table 14 here shows the experimental results of different number of perspectives adopted by the node-graph matching layer of the NGMN model for the classification tasks.

A.7 NGMN WITH DIFFERENT GNNS ON REGRESSION TASKS.

As a supplement to Table 6 in Section 4.3, Table 15 here shows the experimental results of GCN versus GraphSAGE/GIN/GGNN in NGMN for the regression tasks.

Table 14: Regression results of different number of perspectives on **AIDS700** and **LINUX1000**.

Datasets	Model	mse (10^{-3})	ρ	τ	p@10	p@20
AIDS 700	NGMN ($\widetilde{d} = 50$)	**1.133±0.044**	**0.909±0.001**	**0.756±0.002**	**0.487±0.006**	**0.563±0.007**
	NGMN ($\widetilde{d} = 75$)	1.181±0.053	0.905±0.005	0.750±0.007	0.468±0.026	0.547±0.025
	NGMN ($\widetilde{d} = 100$)	1.191±0.048	0.904±0.003	0.749±0.005	0.465±0.011	0.538±0.007
	NGMN ($\widetilde{d} = 125$)	1.235±0.062	0.900±0.007	0.743±0.010	0.456±0.021	0.531±0.014
	NGMN ($\widetilde{d} = 150$)	1.301±0.059	0.893±0.005	0.734±0.007	0.435±0.021	0.511±0.022
LINUX 1000	NGMN ($\widetilde{d} = 50$)	**1.260±0.070**	**0.954±0.004**	**0.829±0.007**	0.825±0.021	0.823±0.025
	NGMN ($\widetilde{d} = 75$)	1.330±0.108	0.952±0.003	0.826±0.006	**0.833±0.029**	**0.843±0.035**
	NGMN ($\widetilde{d} = 100$)	1.561±0.020	0.945±0.002	0.814±0.003	0.743±0.085	0.741±0.086
	NGMN ($\widetilde{d} = 125$)	1.406±0.184	0.950±0.006	0.823±0.015	0.799±0.111	0.803±0.068
	NGMN ($\widetilde{d} = 150$)	1.508±0.083	0.946±0.003	0.815±0.005	0.756±0.033	0.758±0.027

Table 15: Regression results of different GNNs on **AIDS700** and **LINUX1000**.

Datasets	Model	mse (10^{-3})	ρ	τ	p@10	p@20
AIDS 700	NGMN-GCN (Our)	**1.191±0.048**	**0.904±0.003**	**0.749±0.005**	**0.465±0.011**	**0.538±0.007**
	NGMN-(GraphSAGE)	1.275±0.054	0.901±0.006	0.745±0.008	0.448±0.016	0.533±0.014
	NGMN-(GIN)	1.367±0.085	0.889±0.008	0.729±0.010	0.400±0.022	0.492±0.021
	NGMN-(GGNN)	1.870±0.082	0.871±0.004	0.706±0.005	0.388±0.015	0.457±0.017
LINUX 1000	NGMN-GCN (Our)	1.561±0.020	0.945±0.002	0.814±0.003	0.743±0.085	0.741±0.086
	NGMN-GraphSAGE	2.784±0.705	0.915±0.019	0.767±0.028	0.682±0.183	0.693±0.167
	NGMN-GIN	**1.126±0.164**	**0.963±0.006**	**0.858±0.015**	**0.792±0.068**	**0.821±0.035**
	NGMN-GGNN	2.068±0.991	0.938±0.028	0.815±0.055	0.628±0.189	0.654±0.176