Multi-Level Graph Matching Networks for Deep Graph Similarity Learning

Xiang Ling*, Lingfei Wu*, *Member, IEEE,* Saizhuo Wang, Tengfei Ma, Fangli Xu, Alex X. Liu, *Fellow, IEEE,* Chunming Wu, and Shouling Ji, *Member, IEEE,*

***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 the task of graph similarity learning. Recent work on graph similarity learning has 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) framework for computing the graph similarity between any pair of graph-structured objects in an end-to-end fashion. The proposed model MGMN 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 input graphs. Furthermore, to bridge the gap of the lack of standard graph similarity learning benchmarks, we have created and collected a set of datasets for both the graph-graph classification and graph-graph regression tasks with different sizes in order to evaluate the effectiveness and robustness of our models. Comprehensive experiments demonstrate that the proposed model MGMN consistently outperforms state-of-the-art baseline models one both the graph-graph classification and graph-graph regression tasks. Compared with previous work, MGMN also exhibits stronger robustness as the sizes of the two input graphs increase.**

***Index Terms*—Graph similarity, code similarity, deep learning, graph neural network.**

I. Introduction

LEARNING a general similarity metric between arbitrary pairs of graph-structured objects is one of the key challenges in machine learning. Such a learning problem often arises in a variety of real-world applications, ranging from graph similarity searching in graph-based databases [1], to fewshot 3D action recognition [2], unknown malware detection [3] and natural language processing [4] to name just a few. Conceptually, classical *exact* and *error-tolerant* (*i.e.*, inexact) graph matching techniques [5]–[8] provide a strong tool for learning a graph similarity metric. 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 attempt to compute a similarity score between two input graphs. However, in some real-world applications, the constraints of *exact* graph matching techniques are too rigid (*e.g.*, presence of noises or distortions in graphs, neglect of node features, no need for strict one-to-one correspondences, etc.). Thus, in this paper, we focus on the *error-tolerant* graph matching — the graph similarity problem that learns a similarity score between a pair of input graphs. Specifically, we consider the graph similarity problem as to learn a mapping between a pair of input graphs $(G^1, G^2) \in \mathcal{G} \times \mathcal{G}$ and a similarity score $y \in \mathcal{Y}$, based on a set of training triplets $(G_1^1, G_1^2, y_1), \cdots, (G_n^1, G_n^2, y_n) \in \mathcal{G} \times \mathcal{G} \times \mathcal{Y}$ drawn from some fixed but unknown probability distribution in real-world applications.

Recent years have seen a surge of interests in graph neural networks (GNNs), which have been demonstrated to be a powerful class of deep learning models for learning node representations of graph-structured objects [9], [10]. Various GNN models have since been developed for learning effective node embedding vectors for the node classification task [11]–[14], pooling the learned node embedding vectors into a graph-level embedding vector for the general graph classification task [15]–[18], or combining with variational auto-encoder to learn the graph distribution for the graph generation task [19]–[22]. However, there is less study on learning a graph similarity between a pair of input graphs using GNNs.

To learn a graph similarity score between a pair of input graphs, a simple yet straightforward way is to encode each graph as a vector of graph-level embedding via GNNs and then combine the two vectors of both input graphs to make a decision. This approach is useful since it explores graph-level graph-graph interaction features that contain important information of the two input graphs. One obvious limitation of this approach lies in the fact of the ignorance of more fine-grained interaction features among different level embeddings of two input graphs. Very recently, a few attempts of graph matching networks have been made to take into account low-level node-node interactions either by considering the histogram information or spatial patterns (using convolutional neural network [23], *i.e.*, CNN) of the node-wise similarity matrix of node embeddings [24], [25], or by improving the node embeddings of one graph by incorporating the implicit attentive neighbors of the other graph [26].

However, there are two significant challenges making these graph matching networks potentially ineffective: i) how to effectively learn richer cross-level interactions between pairs of input graphs; ii) how to integrate a multi-level granularity (*i.e.*, both cross-level and global-level) of interactions between

Xiang Ling, Saizhuo Wang, Chunming Wu, and Shouling Ji are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: {lingxiang, szwang, wuchunming, sji}@zju.edu.cn); Lingfei Wu and Tengfei Ma are with the IBM T. J. Watson Research Center, NY 10598, USA (e-mail: lwu@email.wm.edu; Tengfei.Ma1@ibm.com); Fangli Xu is with the Squirrel AI Learning, NY, USA (e-mail: lili@yixue.us); Alex X. Liu is with the Ant Financial Services Group, Hangzhou 310013, China (e-mail: alexliu@antfin.com).

Xiang Ling* and Lingfei Wu* contribute equally to this research.

pairs of input graphs for computing the graph similarity in an end-to-end fashion. In order to address the aforementioned challenges, 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-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 leverages the multi-level granularity features that are learned from both cross-level and global-level interactions to perform either the graph-graph classification task or the graph-graph regression task.

The recently proposed graph matching networks [24]–[26] only compute graph similarity scores by considering either the graph-graph classification task (with a binary similarity label $y \in \{-1, 1\}$) [26], or the graph-graph regression task (with a similarity score $y \in (0, 1]$) [24], [25]. It is noted that, the graph-graph classification task here is basically different from the general graph classification task [15], [16] that only assigns each individual graph with a label. In contrast, the graph-graph classification task in our paper learns a binary similarity label (*i.e.*, similar or dissimilar) for *two graphs* instead of *one graph*. To demonstrate the effectiveness of our full model MGMN, we systematically investigate the performance of MGMN compared with these recently proposed graph matching networks on four benchmark datasets for both the graph-graph classification and graph-graph regression tasks.

Another important aspect is previous work does not consider the impact of the size of pairs of input graphs, which often plays an important role in determining the robustness performance of graph matching networks. Motivated by this observation, we consider three different ranges of graph sizes (*i.e.*, [3, 200], [20, 200], and [50, 200]) to evaluate the robustness of models. In addition, to bridge the gap of the lack of standard datasets for the task of graph similarity learning, we create one new dataset from a real-world application together with a previously released dataset by [27] for the graph-graph classification task. 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 first propose a novel node-graph matching network (NGMN) for effectively capturing the rich cross-level interaction features between nodes of a graph and the other whole graph.
- We further present a multi-level graph matching network (MGMN) framework to compute the graph similarity between any pair of graph-structured objects in an end-to-end fashion. In particular, MGMN takes into account both cross-level and graph-level interaction features between a pair of graphs.
- We systematically investigate different factors on the performance of all graph matching networks such as different graph-graph similarity tasks (the graph-graph classification and graph-graph regression tasks) and different sizes of input graphs.
- Comprehensive experiments demonstrate that MGMN consistently outperforms state-of-the-art baselines for both the graph-graph classification and graph-graph regression tasks. Compared with previous work, the proposed MGMN also exhibits stronger robustness as the size of the two input graph increase.

TABLE I
IMPORTANT SYMBOLS AND NOTATIONS

Symbols	Definitions or Descriptions
$G^l,\ l = \{1, 2\}$	The l-th input graph, *i.e.*, G^1 and G^2.
N	The number of nodes in G^1 or the size of G^1.
M	The number of nodes in G^2 or the size of G^2.
$\vec{h}_i^l,\ l = \{1, 2\}$	The hidden embedding of the i-th node in G^l.
$H^l,\ l = \{1, 2\}$	The set of all node embeddings in G^l, *i.e.*, $H^l = \{\vec{h}_i^l\}_{i=1}^{\{N,M\}},\ l = \{1, 2\}$.
$\widetilde{\vec{h}}_G^l,\ l = \{1, 2\}$	The graph-level embedding vector of G^l from NGMN.
$\vec{h}_G^l,\ l = \{1, 2\}$	The graph-level embedding vector of G^l from SGNN.

Roadmap. The remainder of this paper is organized as follows. We briefly introduce the formulation of the graph similarity learning problem in Section II and describe the proposed MGMN model for computing the graph similarity between pairs of graphs in Section III. The performance of the MGMN model is systematically evaluated and analyzed in Section IV. Section V surveys related work and Section VI finally concludes this work.

II. PROBLEM FORMULATION

In this section, we briefly introduce the problem formulation as follows. 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 $y = s(G^1, G^2) \in \mathcal{Y}$. 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 $(v_i, v_{i'}) \in \mathcal{E}^1$ (binary or weighted) formulating an adjacency matrix $A^1 \in \mathcal{R}^{N \times N}$, and a degree matrix $\widetilde{D}_{ii}^1 = \sum_{i'} A_{ii'}^1$. Similarly, the graph $G^2 = (\mathcal{V}^2, \mathcal{E}^2)$ is represented as a set of M nodes $v_j \in \mathcal{V}^2$ with a feature matrix $X^2 \in \mathcal{R}^{M \times d}$, edges $(v_j, v_{j'}) \in \mathcal{E}^2$ (binary or weighted) formulating an adjacency matrix $A^2 \in \mathcal{R}^{M \times M}$, and a degree matrix $\widetilde{D}_{jj'}^2 = \sum_{j'} A_{jj'}^2$. It is noted that, when performing the graph-graph classification task, y is a binary similarity label $y \in \mathcal{Y} = \{-1, 1\}$; when performing the graph-graph regression task, y is the continuous similarity score $y \in \mathcal{Y} = (0, 1]$. We train our models based on a set of training triplet of input graph pairs and a scalar output score $(G_1^1, G_1^2, y_1), \cdots, (G_n^1, G_n^2, y_n) \in \mathcal{G} \times \mathcal{G} \times \mathcal{Y}$ drawn from some fixed but unknown probability distribution in real-world applications. A summary of important symbols and notations used in this paper can be found in Table I.

III. MULTI-LEVEL GRAPH MATCHING NETWORKS

In this section, we detail the proposed multi-level graph matching network (MGMN) framework, which consists of both a node-graph matching network (NGMN) and a siamese graph neural network (SGNN). The overall model architecture of MGMN is shown in Fig. 1. In the following subsections,

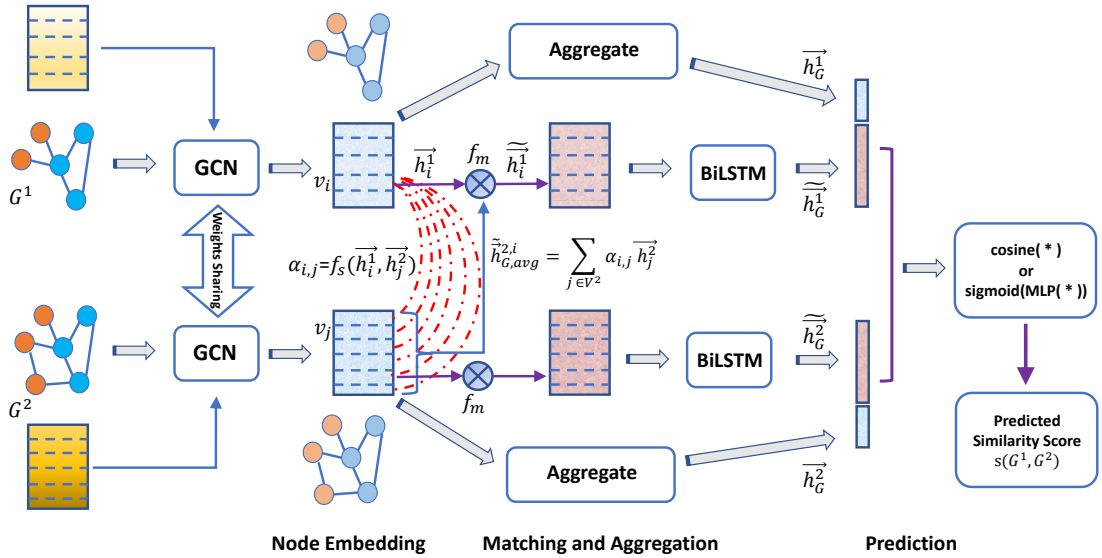


Fig. 1. Overview architecture of the full model MGMN, consisting of two partial models: SGNN and NGMN. The two input graphs first go through NGMN and SGNN, which results in aggregated graph-level embedding vectors after their corresponding aggregation layers. After that, we concatenate the two aggregated graph-level embedding vectors for each graph G^l, where one vector $\overrightarrow{\widetilde{h}}^l_G$ (long & pink) is from NGMN and another vector $\vec{h}^l_G$ (short & blue) is from SGNN, and then the two concatenated embedding vectors into the following prediction layer.

we first introduce NGMN for effectively learning the cross-level node-graph interaction features between nodes of one graph and the other whole graph, and then outline SGNN for learning the global-level interaction features between the two graphs. Finally, we present our full model MGMN that combines NGMN and SGNN to learn both cross-level node-graph interactions as well as graph-level graph-graph interactions.

A. 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 input 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. In general, NGMN consists of four layers: 1) node embedding layer; 2) node-graph matching layer; 3) aggregation layer; and 4) prediction layer. We will illustrate each layer in detail as follows.

1) Node Embedding Layer: The siamese network architecture [28] has achieved great success in many metric learning tasks such as visual recognition [29], [30], video segmentation [31], [32] and sentence similarity analysis [33], [34]. In this layer, we consider a multi-layer graph convolution network (GCN) [11] with the siamese network architecture to generate node embeddings $H^l = \{\vec{h}^l_i\}_{i=1}^{\{N,M\}} \in \mathcal{R}^{\{N,M\}\times d'}$ for all nodes in either G^1 or G^2 as follows.

$$H^l = \sigma\Big(\bar{A}^l \ldots \sigma\Big(\bar{A}^l\, \sigma\Big(\bar{A}^l X^l W^{(0)}\Big) W^{(1)}\Big) \ldots W^{(T-1)}\Big),\ l = \{1,2\} \quad (1)$$

Here, 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; N and M denote the number of nodes for both G^1 and G^2; H^l is the set of all learned node embeddings of the graph G^l; $l = \{1,2\}$ in the superscript of H^l, $\bar{A}^l$, X^l indicates it belongs to G^1 or G^2; T is the number of GCN layers; $W^{(t)}, t \in \{0,1,\ldots,T-1\}$ is the layer-specific trainable weighted matrix of the t-th GCN layer. It is noted that the siamese network architecture shares the parameters of GCN when training on pairs of input graphs (G^1, G^2), and the number of GCN layers required depends on specific real-world applications.

2) Node-Graph Matching Layer: This layer is the key part of our NGMN model, which can effectively learn the cross-level interactions between nodes of a graph and the other whole graph. There are generally two steps for this layer: i) calculate the graph-level embedding vector of a graph; ii) compare the node embeddings of a graph with the associated graph-level embedding vector of the other whole graph and then produce a similarity feature vector.

A simple approach to obtain the graph-level embedding vector of a graph is to directly perform pooling operations, *e.g.*, element-wise max pooling. However, this approach does not consider any information from the node embeddings that the resulting graph-level embedding vector will compare with later. To build more tight and informative interactions between the two graphs for learning the graph-level embedding vector of each other, we first calculate a cross-graph attention coefficient $\alpha_{i,j}$ 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 coefficient $\beta_{j,i}$ 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. To be specific, these two cross-graph attention coefficients can be computed with an attention function f_s independently,

$$\begin{aligned} \alpha_{i,j} &= f_s(\vec{h}^1_i, \vec{h}^2_j) = \text{cosine}(\vec{h}^1_i, \vec{h}^2_j), \quad v_j \in \mathcal{V}^2 \\ \beta_{j,i} &= f_s(\vec{h}^2_j, \vec{h}^1_i) = \text{cosine}(\vec{h}^2_j, \vec{h}^1_i), \quad v_i \in \mathcal{V}^1 \end{aligned} \quad (2)$$

where f_s is the attention function for computing the similarity

score between two node embedding vectors. For simplicity, we use the 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 corresponding attentive graph-level embedding vector of another graph. Specifically, from the view of the node $v_i \in \mathcal{V}^1$ in G^1, we compute the attentive graph-level embedding vector $\widetilde{\vec{h}}^{2,i}_{G,avg}$ of G^2 by weighted averaging all node embeddings of G^2 with corresponding attentions. Likewise, $\widetilde{\vec{h}}^{1,j}_{G,avg}$ is computed as the attentive graph-level embedding vector of G^1 from the view of the node $v_j \in \mathcal{V}^2$ in G^2. Thus, we compute these two attentive graph-level embeddings as follows.

$$\begin{aligned} \widetilde{\vec{h}}^{2,i}_{G,avg} &= \sum_{j \in \mathcal{V}^2} \alpha_{i,j} \vec{h}^2_j, \quad v_i \in \mathcal{V}^1 \\ \widetilde{\vec{h}}^{1,j}_{G,avg} &= \sum_{i \in \mathcal{V}^1} \beta_{j,i} \vec{h}^1_i, \quad v_j \in \mathcal{V}^2 \end{aligned} \tag{3}$$

Next, we define a multi-perspective matching function f_m to compute the similarity feature vector by comparing two input vectors of $\vec{x}_1$ and $\vec{x}_2$.

$$\begin{aligned} \widetilde{\vec{h}}[k] &= f_m(\vec{x}_1, \vec{x}_2, \vec{w}_k) \\ &= \text{cosine}(\vec{x}_1 \odot \vec{w}_k, \vec{x}_2 \odot \vec{w}_k), \quad k = 1, \ldots, \widetilde{d} \end{aligned} \tag{4}$$

where $\odot$ is the element-wise multiplication operation, $\widetilde{\vec{h}}[k] \in \mathcal{R}$ denotes the output similarity score in terms of k-th perspective, and $\vec{w}_k \in \mathcal{R}^{d'}$ represents the learnable weight vector in the k-th perspective. When considering a total of $\widetilde{d}$ number of perspectives for the multi-perspective matching function f_m, the trainable weighted matrix will be $W_m = \{\vec{w}_k\}^{\widetilde{d}}_{k=1} \in \mathcal{R}^{d' \times \widetilde{d}}$. After that, we will obtain a $\widetilde{d}$-dimension vector of similarity features, *i.e.*, $\widetilde{\vec{h}} \in \mathcal{R}^{\widetilde{d}}$.

It is worth noting that the proposed f_m essentially shares a similar spirit with the multi-head attention mechanism [35]. However, the most significant difference is that, the multi-head attention mechanism employs $\widetilde{d}$ number of trainable weighted *matrices*, while f_m uses $\widetilde{d}$ number of trainable weighted *vectors* instead. It is obvious that our methods uses substantially fewer training parameters, which may reduce potential over-fitting as well as significantly speed up our computation.

With the defined multi-perspective matching function f_m in Equation (4), we use it to compare the i-th node embedding in graph G^1 with the corresponding attentive graph-level embedding $\widetilde{\vec{h}}^{2,i}_{G,avg}$ of the other graph G^2. The resulting similarity feature vector $\vec{h}^1_i \in \mathcal{R}^{\widetilde{d}}$ is thus considered as the updated node embedding of i-th node in graph G^1. Similarly, we also use f_m to compare the j-th node embedding in the graph G^2 with the attentive graph-level embedding $\widetilde{\vec{h}}^{1,j}_{G,avg}$ of graph G^1, and consider the resulting feature vector $\widetilde{\vec{h}}^2_i \in \mathcal{R}^{\widetilde{d}}$ as the update node embedding of j-th node in the graph G^2. Specifically, for all nodes in both G^1 and G^2, their corresponding node-graph interaction features can thus be computed by,

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

After performing the above node-graph matching layer over all nodes for both graphs, these newly generated interaction features of nodes are considered and collected as the new feature matrices for G^1 and G^2, *i.e.*, $\widetilde{H}^1 = \{\widetilde{\vec{h}}^1_i\}^N_{i=1} \in \mathcal{R}^{N \times \widetilde{d}}$ and $\widetilde{H}^2 = \{\widetilde{\vec{h}}^2_j\}^M_{j=1} \in \mathcal{R}^{M \times \widetilde{d}}$, which capture the cross-level interaction features between node embeddings of a graph and a corresponding graph-level embedding of the other graph.

3) Aggregation Layer: To aggregate the learned interactions from the node-graph matching layer, we employ the bidirectional LSTM (*i.e.*, BiLSTM) [36], [37] to aggregate an unordered set of node embeddings for each graph as follows.

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

Here, BiLSTM in Equation (6) takes a random permutation of the node embeddings as the input and concatenate the two last hidden vectors from both directions (*i.e.*, forward and backward) of the bidirectional LSTM as the representation of each graph. The resulting $\widetilde{\vec{h}}^l_G \in \mathcal{R}^{2\widetilde{d}}$ represents the aggregated graph-level embedding vector for the graph G^1 or G^2.

Ideally, an aggregator function would be invariant to permutations of its input while maintaining a large expressive capacity. However, we take BiLSTM as the default aggregation function, which is not permutation invariant on the set of node embeddings. The reason is two-fold. First, LSTM-related aggregators have been employed in previous work [12], [38] due to their larger expressive model capability. Second, we conduct extensive experiments on the choice of aggregators in NGMN and show that BiLSTM achieves consistently better performance than other aggregator functions (*e.g.*, max pooling aggregator, see Table III and Table IV in Section IV-D for detailed comparison results).

4) Prediction Layer: After the aggregated graph-level embedding vectors $\widetilde{\vec{h}}^1_G$ and $\widetilde{\vec{h}}^2_G$ are obtained, we then use them to compute a similarity score $s(G^1, G^2)$ between G^1 and G^2. Depending on the specific tasks, *i.e.*, the graph-graph classification task and the graph-graph regression task, we have slightly different ways to calculate the final predicted similarity score.

For the *graph-graph classification* task, we directly compute the cosine similarity of two graph-level embedding vectors as follows, as it is quite common to employ the cosine similarity in other classification tasks [27], [39].

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

Differently, the predicted result of the graph-graph regression task is continuous and is normalized in a range of (0,1]. Thus, for the *graph-graph regression* task, we first concatenate the two graph-level embedding vector into $\big[\widetilde{\vec{h}}^1_G, \widetilde{\vec{h}}^2_G\big]$ and then employ four standard fully connected layers to gradually project the dimension of resulting vector down to a scalar

of the dimension 1. Since the expected similarity score $\widetilde{y}$ should be in the range of (0, 1], we perform the sigmoid activation function to enforce the similarity score in this range. We therefore compute the similarity score for the graph-graph regression task as follows.

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

where $\big[\cdot\ ;\ \cdot\big]$ denotes the concatenation operation over two input vectors and MLP denotes the employed four fully connected layers.

5) Model Training: The model is trained on a set of n training triplets of two input graph-structured objects and a scalar output score $(G_1^1, G_1^2, y_1), ..., (G_n^1, G_n^2, y_n) \in \mathcal{G} \times \mathcal{G} \times \mathcal{Y}$. For both the graph-graph classification and graph-graph regression tasks, we train the models with the loss function of mean square error to compare the computed similarity score $\widetilde{y}$ with the ground-truth similarity score y.

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(\widetilde{y} - y)^2 \tag{9}$$

B. SGNN for Global-Level Interaction Learning

The graph-level embeddings contain important information of a graph. Therefore, learning graph-level interaction features between two graphs could be an important supplementary component for learning the graph similarity between two graphs. In order to capture the global-level interaction features between two graphs, we present the siamese graph neural network (SGNN) which is also based on siamese network architecture as presented in the previous Section III-A1. To be specifically, our SGNN consists of three layers: 1) node embedding layer; 2) aggregation layer; 3) prediction layer. We detail each layer of SGNN in the following.

1) Node Embedding Layer: For the sake of simplicity, we also adopt a multi-layer GCN with the siamese network architecture to generate context embeddings for all nodes, *i.e.*, $H^1 = \{\vec{h}_i^1\}_{i=1}^N \in \mathcal{R}^{N\times d'}$ and $H^2 = \{\vec{h}_j^2\}_{j=1}^M \in \mathcal{R}^{M\times d'}$, for both graphs G^1 and G^2, respectively. This is the same as the node embedding layer of NGMN that has already been explored in Section III-A1.

Conceptually, the node embedding layer in SGNN could be chosen to be an independent or shared with the node embedding layer in NGMN. As shown in Fig. 1, our SGNN model shares the same node embedding layer with NGMN due to two reasons: i) sharing the GCN parameters in the node embedding layer means reducing the number of parameters by half, which helps mitigate possible over-fitting; ii) the shared GCN models maintain the consistency of the resulting node embeddings for both NGMN and SGNN, potentially leading to more aligned cross-level and graph-level interaction features. After all the node embeddings H^1 and H^2 for two input graphs have been computed, they will be fed into the subsequent aggregation layer.

2) Aggregation Layer: With the computed node embeddings H^1 and H^2 for both G^1 and G^2, we need to aggregate them to formulate their corresponding graph-level embedding vectors $\vec{h}_G^1$ and $\vec{h}_G^2$ as follows.

$$\vec{h}_G^l = \text{Aggregation}\Big(\{\vec{h}_i^l\}_{i=1}^{\{N,M\}}\Big), \quad l = \{1, 2\} \tag{10}$$

where Aggregation represents the aggregation function that outputs a corresponding graph-level vector. Without a doubt, we can use the BiLSTM aggregator function that has been introduced in Section III-A3. In addition to BiLSTM, we would like to employ other *simpler* aggregator functions, such as element-wise max pooling (Max) and element-wise max pooling following a transformation by applying a standard fully connected layer (FCMax).

3) Prediction Layer: After the aggregated graph-level embedding vectors $\vec{h}_G^1$ and $\vec{h}_G^2$ are obtained, we then use these two graph embeddings to compute the similarity score of $s(G^1, G^2)$. Just like the prediction layer in NGMN, we use Equation (7) and Equation (8) to predict the similarity score for the graph-graph classification task and the graph-graph regression task, respectively. We also use the same loss function of mean square error in Equation (9) to train the SGNN model. In this way, we can also easily compare the performance difference between SGNN and NGMN.

Compared with other graph-graph interactions. Although SimGNN [24] also learns graph-graph interaction features, our SGNN is still different from it in three aspects. i) We apply a siamese network architecture with one shared multi-layer GCN model to learn node embeddings rather than two independent multi-layer GCN models; ii) SGNN only employs a simple aggregation function to learn a graph-level embedding vector while SimGNN uses a context-aware attention method; iii) SGNN directly employs the cosine function or the concatenation with fully connected layers to learn the graph-graph interaction features, whereas SimGNN uses a more sophisticated neural tensor network [40] to capture the graph-graph interaction features.

C. Discussions on Our Full Model — MGMN

1) 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 for better representation learning in computing the graph similarity between two input graphs. As shown in Fig. 1, for the full model MGMN, the two input graphs G^l (l = {1, 2}) first go through NGMN and SGNN, which produce two corresponding graph-level embedding vectors, *i.e.*, $\widetilde{\vec{h}}_G^l$ and $\vec{h}_G^l$, respectively. After that, we concatenate the two aggregated graph-level embedding vectors from NGMN and SGNN for each graph, and then feed those concatenated embedding into the following prediction prediction layer as presented in Section III-A4. In particular, we compute the similarity score $s(G^1, G^2)$ between G^1 and G^2 for both the graph-graph classification and graph-graph regression tasks as follows.

$$\widetilde{y} = s(G^1, G^2) = \text{cosine}(\big[\widetilde{\vec{h}}_G^1; \vec{h}_G^1\big], \big[\widetilde{\vec{h}}_G^2; \vec{h}_G^2\big]) \tag{11}$$

$$\widetilde{y} = s(G^1, G^2) = \text{sigmoid}\Big(\text{MLP}\Big(\Big[\big[\widetilde{\vec{h}}_G^1; \vec{h}_G^1\big]; \big[\widetilde{\vec{h}}_G^2; \vec{h}_G^2\big]\Big]\Big)\Big) \tag{12}$$

TABLE II
SUMMARY STATISTICS OF DATASETS FOR BOTH THE GRAPH-GRAPH CLASSIFICATION TASK & THE GRAPH-GRAPH REGRESSION TASK.

Tasks	Datasets	Sub-Datasets	# of Graphs	# of Functions	AVG # of Nodes	AVG # of Edges	Initial Feature Dimensions
Graph-Graph Classification Task	**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	
Graph-Graph Regression Task	**AIDS700**	-	700	-	8.90	8.80	29
	LINUX1000	-	1,000	-	7.58	6.94	1

2) Complexity Analysis: 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 Equation (3) and Equation (5). Compared to recently proposed work [24]–[26], their computational complexities are highly comparable.

IV. EXPERIMENT

In this section, we systematically evaluate the performance of our full model MGMN compared with recently proposed baseline models on four benchmark datasets for both the graph-graph classification and graph-graph regression tasks. In particular, we first introduce the benchmark datasets of both tasks in Section IV-A, provide details of the experimental settings in Section IV-B, describe the baseline models to be compared with our models in Section IV-C, and finally present the main evaluation results as well as conduct ablation studies in Section IV-D and Section IV-E, respectively.

A. Tasks & Datasets

1) Graph-Graph Classification Task & Datasets: We evaluate our models on the graph-graph classification task of computing a similarity score (*i.e.*, $y \in \{-1, 1\}$) between two binary functions, which is the heart of many binary security problems [26], [27], [41]. Conceptually, two binary functions that are compiled from the same source code but under different settings (*e.g.*, architectures, compilers, optimization levels, etc) are considered to be semantically similar to each other. To learn the similarity score from a pair of binary functions, we represent each binary function with a control flow graph (CFG), 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, computing a similarity score between two binary functions can be cast as the problem of learning the similarity score $s(G^1, G^2)$ between two CFGs 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**[1] and **OpenSSL**[2], to evaluate our models on the graph-graph classification task. More details about how we prepare both datasets can be found in Appendix A.

Besides, existing work does not consider the impact of the graph size on the performance of graph matching networks. However, we find the larger the graph size is, the worse the performance is. Therefore, it is important to evaluate the robustness of graph matching networks in this setting. We thus further split each dataset into 3 sub-datasets (*i.e.*, [3, 200], [20,200], and [50,200]) according to the size ranges of pairs of input graphs.

It is noted that, although there are many benchmark datasets [42] for the general graph classification task [15], [16], these datasets cannot be directly employed in our graph-graph classification task as we cannot treat the two input graphs with the same labels as "similar". This is because the general graph classification task only assigns each graph with a label, while our graph-graph classification task learns a binary similarity label (*i.e.*, similar or dissimilar) for pairs of two graphs instead of one graph.

2) Graph-Graph Regression Task & Datasets: We evaluate our models on learning the graph edit distance (GED) [43], [44], which measures the structural similarity between two input graphs. Formally, GED is defined as the cost of the least sequence of edits that transform one graph into another graph, where an edit can be an insertion/deletion of a node/edge. Instead of directly computing the GED between two G^1 and G^2, we try to learn a similarity score $s(G^1, G^2)$, which is the normalized exponential of GED, $exp^{-\left[\frac{GED(G^1,G^2)}{(N+M)/2}\right]}$, in the range of $(0, 1]$. In particular, we employ two benchmark datasets, *i.e.*, **AIDS700** and **LINUX1000**, which are released by [24] and publicly available.[3] Each dataset contains a set of pairs of input graphs as well as their ground-truth GED scores, which are computed by exponential-time exact GED computation algorithm A^* [45], [46]. As the ground-truth GEDs of another dataset IMDB-MULTI are provided with *inexact* approximations, we thus do not consider this dataset in our evaluation.

In summary, for both the graph-graph classification and graph-graph regression tasks, we follow the same training/validation/testing split as previous work [24], [26] for fair comparisons. For the graph-graph classification task, Bai et al. split each dataset into three disjoint subsets of binary functions in which 80% for training, 10% for validation and 10% for

[1]https://www.ffmpeg.org/
[2]https://www.openssl.org/
[3]https://github.com/yunshengb/SimGNN

testing. For each dataset in the graph-graph regression task, Li et al. randomly split 60%, 20%, and 20% of all graphs as the training/validation/testing subsets, and then build the pairwise datasets. Table II shows the summary statistic for all datasets.

B. Implementation Settings

To set up our models, including SGNN, NGMN, and MGMN, we use a three-layer GCN in the node embedding layer and each of the GCN layer 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 aggregation layer of SGNN, we can employ different aggregation functions (*i.e.*, Max, FCMax, and BiLSTM) as introduced previously in Section III-B. For NGMN, we set the number of perspectives $\widetilde{d}$ to 100. 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 two last hidden vectors of both directions of BiLSTM, which results in a 200-dimension vector as the graph-level embedding vector.

Our implementation is built using PyTorch [47] and PyTorch Geometric [48]. To train our models, we use the Adam optimizer [49]. For the graph-graph classification task, 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 set, 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 CFGs that are compiled from the same source function as G, and G^{neg} is selected from the other graphs. By default, each batch includes 5 positive and 5 negative pairs. For the graph-graph regression task, 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 normalized GED between G^1 and G^2. Noted that all experiments are conducted on a PC equipped with 8 Intel Xeon 2.2GHz CPU and one NVIDIA GTX 1080 Ti GPU.

C. Baseline Methods

To evaluate the effectiveness of our models, we consider 3 state-of-the-art baseline models for comparisons as follows.

1) *SimGNN* [24] adopts a multi-layer GCN and employs two strategies to calculate the GED between two graphs: one uses a neural tensor network to capture the graph-graph interactions; another uses histogram features extracted from two sets of node embeddings;
2) *GMN* [26] 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;
3) *GraphSim* [25] extends SimGNN by turning the two sets of node embeddings into a similarity matrix and then processing the matrix with CNNs.

As the three baselines only consider either the graph-graph classification task or the graph-graph regression task, we slightly adjust the last layer of the model or the loss function of each baseline to make a fair comparison on both tasks. Detailed experimental settings of these baselines are given in Appendix B. It is also worth noting that we repeat all experiments 5 times and report the mean and standard deviation of experimental results, with the best in **bold**.

D. Evaluation Results the Proposed Models

In this subsection, we compare our proposed models with all three state-of-the-art baseline models on both the graph-graph classification and graph-graph regression tasks. Specifically, our models contain a full model **MGMN** and two partial models, *i.e.*, SGNN and NGMN. For both SGNN and NGMN, we employ three different aggregator functions (*i.e.*, Max, FCMax, and BiLSTM) in their corresponding aggregation layers. As the full model **MGMN** combines both SGNN and NGMN, **MGMN** must have two aggregation functions for SGNN and NGMN. In the following subsection, we use the bracket after the model indicates the employed aggregator function(s). For instance, SGNN (Max) represents the SGNN model with the Max aggregator and NGMN (BiLSTM) denotes the NGMN model with the BiLSTM aggregator; **MGMN** (FCMax + BiLSTM) means that **MGMN** takes FCMax as the aggregator for SGNN and BiLSTM as the aggregator for NGMN.

1) Graph-Graph Classification Task: For the graph-graph classification task of detecting whether two binary functions are similar or not, we measure the Area Under the ROC Curve (AUC) [50] for classifying pairs of input graphs in the same testing dataset. AUC is one of the most commonly used evaluation metrics to evaluate the binary classification models as AUC is independent of the manually selected threshold. Apparently, the larger AUC, the better performance of the models. The main results are shown in Table III.

We first investigate the impact of different aggregation functions employed by our models, including SGNN, NGMN, and **MGMN**. For SGNN and NGMN, it is clearly seen from Table III that the Max aggregator achieves the worst performance and the BiLSTM aggregator offers better performance on both **FFmpeg** and **OpenSSL** datasets. One possible reason we conjecture is that BiLSTM might admit more expressive ability as the aggregation model. In addition, when comparing the results of NGMN (BiLSTM) with SGNN (BiLSTM), NGMN (BiLSTM) offers consistently superior performance on almost all sub-datasets, demonstrating the benefits of the node-graph matching mechanism that captures the cross-level interactions between two graphs.

As NGMN (BiLSTM) shows the best performance, for the full model **MGMN**, we fix BiLSTM as the default aggregator for the NGMN part and vary the aggregator (*e.g.*, Max, FCMax, or BiLSTM) for the SGNN part. Therefore, compared **MGMN** with the two partial models (*i.e.*, SGNN and NGMN), it is observed that **MGMN** further improves the performance of NGMN together with the graph-level interaction features learned from SGNN. This confirms that SGNN and NGMN are two indispensable partial models for our full model **MGMN**.

Besides, we also compare our full model **MGMN** with 3 baseline models. The experimental results that our models with the BiLSTM aggregator clearly achieve the state-of-the-art performance on all 6 sub-datasets for both **FFmpeg** and

TABLE III
SUMMARY OF EXPERIMENTAL RESULTS ON THE GRAPH-GRAPH CLASSIFICATION TASK 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 (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 (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
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 (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
MGMN (Max + BiLSTM)	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 + BiLSTM)	**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 + 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

TABLE IV
SUMMARY OF EXPERIMENTAL RESULTS ON THE GRAPH-GRAPH REGRESSION TASK IN TERMS OF mse, ρ, τ, $p@10$ & $p@20$.

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 (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 (BiLSTM)	1.422±0.044	0.881±0.005	0.718±0.006	0.376±0.020	0.472±0.014
	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 (BiLSTM)	1.191±0.048	0.904±0.003	0.749±0.005	0.465±0.011	0.538±0.007
	MGMN (Max + BiLSTM)	1.210±0.020	0.900±0.002	0.743±0.003	0.461±0.012	0.534±0.009
	MGMN (FCMax + BiLSTM)	1.205±0.039	0.904±0.002	0.749±0.003	0.457±0.014	0.532±0.016
	MGMN (BiLSTM + BiLSTM)	**1.169±0.036**	**0.905±0.002**	**0.751±0.003**	0.456±0.019	0.539±0.018
LINUX1000	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 (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 (BiLSTM)	2.140±1.668	0.935±0.050	0.825±0.100	0.878±0.012	0.865±0.007
	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 (BiLSTM)	1.561±0.020	0.945±0.002	0.814±0.003	0.743±0.085	0.741±0.086
	MGMN (Max + BiLSTM)	1.054±0.086	0.962±0.003	0.850±0.008	0.877±0.054	0.883±0.047
	MGMN (FCMax + BiLSTM)	1.575±0.627	0.946±0.019	0.817±0.034	0.807±0.117	0.784±0.108
	MGMN (BiLSTM + BiLSTM)	**0.439±0.143**	**0.985±0.005**	0.919±0.016	0.955±0.011	**0.943±0.014**

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

OpenSSL datasets. Particularly when the graph size increases, both **MGMN** and the key component NGMN (BiLSTM) models show better performance and robustness than state-of-the-art methods.

2) Graph-Graph Regression Task: For the graph-graph regression task of computing the similarity score GED between two input graphs, we measure the Mean Square Error (mse), Spearman's Rank Correlation Coefficient (ρ) [51], Kendall's Rank Correlation Coefficient (τ) [52], and precision at k ($p@k$) as previous work [24], [25] for fair comparisons. Apparently, the smaller mse, the better performance of models. But for ρ, τ, and $p@k$, the larger the better. All results of both datasets are summarized in Table IV.

For the impact of different aggregator functions employed by both SGNN and NGMN, our experimental results draw similar conclusions that the BiLSTM aggregator offers superior performance on both datasets and NGMN (BiLSTM) achieves better performance than SGNN (BiLSTM) in terms of most evaluation metrics. Furthermore, compared **MGMN** with the two partial models (*i.e.*, SGNN and NGMN), it is clearly seen that **MGMN** further improves the performance of either SGNN or NGMN. These observations confirm again that NGMN and SGNN are two indispensable parts for the full model **MGMN**, which could capture both the cross-level node-graph interactions and global-level graph-graph interactions for better representation learning in computing the graph

TABLE V
THE GRAPH-GRAPH 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 (# of Heads = 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

similarity between two graphs.

We also compare our full model **MGMN** with state-of-the-art baseline models on the graph-graph regression task. From Table IV, it can be observed that, although GraphSim shows better performance than the other two baselines, our full model **MGMN** and its key component NGMN outperform all baselines on both **AIDS700** and **LINUX1000** datasets in terms of most evaluation metrics.

E. Ablation Studies on NGMN

In this subsection, we explore ablation studies to measure the contributions of different components in NGMN, which is the key partial model of the full model **MGMN**. The reasons why we conduct ablation studies on NGMN instead of **MGMN** are as follows. First, we have already shown the performance of SGNN, NGMN, and **MGMN** in the above Section IV-D, highlighting the importance of our proposed cross-level interactions learned from NGMN and graph-level interactions learned from SGNN. Second, conducting only ablation studies on NGMN would make the evaluation results easier to be observed and clearer to be compared, as it excludes the potential influences of SGNN.

1) Impact of Different Attention Functions: As explored in Section III-A2, the proposed multi-perspective matching function shares similar spirits with the multi-head attention mechanism [35], which makes it interesting to compare them. Therefore, we investigate the impact of these two different mechanisms for NGMN with classification results shown in Table V. In our evaluation, the number of heads K in the multi-head attention model is set to 6 because of the substantial consumption of resources of multi-head attention computations. Even though, the number of parameters of multi-head attention is still times more than our multi-perspective matching function. Interestingly, it is observed that our proposed multi-perspective matching function consistently outperforms the results of the multi-head attention by quite a large margin. We suspect that the multi-perspective matching function uses attention weighted vectors rather than matrices, which may reduce the potential over-fitting.

2) Impact of Different Numbers of Perspectives: We further investigate the impact of different numbers of perspectives adopted by the node-graph matching layer in NGMN. 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 VI, it is clearly seen that the AUC score of NGMN does not increase as the number of perspectives grows for the graph-graph classification task. Similar results of the graph-graph regression task can be found in Table IX in the Appendix. We thus conclude that the performance of NGMN is not sensitive to the number of perspectives $\widetilde{d}$ (from 50 to 150) and we make $\widetilde{d} = 100$ by default.

3) Impact of Different GNN Variants: We investigate the impact of different GNN variants including GraphSAGE [12], GIN [53], and GGNN [54] adopted by the node embedding layer of NGMN for both the graph-graph classification and graph-graph regression tasks. Table VII presents the results of the graph-graph classification task (see Table X in the Appendix for the results of the graph-graph regression task). In general, the performance of different GNNs is quite similar for all datasets of both tasks, which indicates that NGMN is not sensitive to the choice of GNN variants 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 models can be further improved by adopting more advanced GNN models or choosing the most appropriate GNNs according to different real-world application tasks.

4) Impact of Different Numbers of GCN Layers: We also examine how the number of GNN (*i.e.*, GCN) layers would affect the performance of our models for both the graph-graph classification and graph-graph regression tasks. Follow the same default experimental settings, 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 VIII for the graph-graph classification task as well as Table XI in the Appendix for the graph-graph regression task.

It can be observed from Table VIII 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 fewer 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 datasets; NGMN with 2-layer runs poorly on the [20, 200] and [50, 200] sub-datasets of **FFmpeg** as well as the [50, 200] sub-dataset of **OpenSSL**.

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

TABLE VI
THE GRAPH-GRAPH CLASSIFICATION RESULTS OF NGMN MODELS WITH 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**

TABLE VII
THE GRAPH-GRAPH CLASSIFICATION RESULTS OF NGMN MODELS WITH 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**

TABLE VIII
THE GRAPH-GRAPH CLASSIFICATION RESULTS OF NGMN MODELS WITH DIFFERENT NUMBERS OF GCN LAYERS IN TERMS OF AUC SCORES (%).

Model	FFmpeg			OpenSSL		
	[3, 200]	[20, 200]	[50, 200]	[3, 200]	[20, 200]	[50, 200]
NGMN-(1 layer)	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**

V. RELATED WORK

A. Conventional Graph Matching

As introduced in Section I, the general graph matching can be categorized into *exact* and *error-tolerant* (*i.e.*, inexact) graph matching techniques. Specifically, exact graph matching techniques aim to find a strict one-to-one correspondence between two (in large parts) identical graphs being matched, while error-tolerant graph matching techniques allow matching between completely non-identical graphs [44]. In real-world applications, the constraint of exact graph matching is too rigid such as the presence of noises or distortion in graphs, neglect of node features, and so on. Therefore, an amount of work has been proposed to solve the error-tolerant graph matching problem, which is usually quantified by specific similarity metrics, such as GED [43], [44], maximum common subgraph [55], or even more coarse binary similarity, according to different real-world applications. Particularly for the computation of GED, it is a well-studied NP-hard problem and suffers from exponential computational complexity and huge memory requirements for exact solutions in practice [56]–[58].

B. Graph Similarity Computation

Considering the great significance and challenge of computing the graph similarity between pairs of graphs, a popular line of research of graph matching techniques focuses on developing approximation methods for better accuracy and efficiency, including traditional heuristic methods [43], [44], [59], [60] and recent data-driven graph matching networks [24]–[26], as detailed in Section IV-C.

Our work belongs to the data-driven 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 full model MGMN successfully captures multi-level richer interactions between two graphs; 2) our work is the first one to systematically evaluate the performance on both the graph-graph classification and graph-graph regression tasks as well as the size of input graphs.

VI. 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 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 the graph-graph classification and graph-graph regression tasks. One interesting future direction is to adapt our model MGMN for solving different real-world applications such as malware detection, text entailment, and question answering with knowledge graphs.

REFERENCES

[1] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proceedings of IEEE International Conference on Data Mining*. IEEE, 2002, pp. 721–724.

[2] M. Guo, E. Chou, D.-A. Huang, S. Song, S. Yeung, and L. Fei-Fei, "Neural graph matching networks for fewshot 3d action recognition," in *ECCV*, 2018, pp. 653–669.

[3] S. Wang, Z. Chen, X. Yu, D. Li, J. Ni, L.-A. Tang, J. Gui, Z. Li, H. Chen, and P. S. Yu, "Heterogeneous graph matching networks for unknown malware detection," in *IJCAI*, 2019.

[4] Y. Chen, L. Wu, and M. J. Zaki, "Reinforcement learning based graph-to-sequence model for natural question generation," in *ICLR*, 2020.

[5] H. Bunke and G. Allermann, "Inexact graph matching for structural pattern recognition," *Pattern Recognition Letters*, vol. 1, no. 4, pp. 245–253, 1983.

[6] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola, "Learning graph matching," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 6, pp. 1048–1058, 2009.

[7] K. Riesen, X. Jiang, and H. Bunke, "Exact and inexact graph matching: Methodology and applications," in *Managing and Mining Graph Data*. Springer, 2010, pp. 217–247.

[8] S. P. Dwivedi and R. S. Singh, "Error-tolerant graph matching using node contraction," *Pattern Recognition Letters*, vol. 116, pp. 58–64, 2018.

[9] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[12] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[13] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[14] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[15] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018.

[16] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *KDD*, 2019.

[17] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," *arXiv preprint arXiv:1904.08082*, 2019.

[18] H. Gao and S. Ji, "Graph u-nets," *arXiv:1905.05178*, 2019.

[19] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," *arXiv preprint arXiv:1802.03480*, 2018.

[20] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[21] B. Samanta, A. De, N. Ganguly, and M. Gomez-Rodriguez, "Designing random graph models using variational autoencoders with applications to chemical design," *arXiv preprint arXiv:1802.05283*, 2018.

[22] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *ICML*, 2018.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[24] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang, "SimGNN: A neural network approach to fast graph similarity computation," in *WSDM*. ACM, 2019, pp. 384–392.

[25] Y. Bai, H. Ding, K. Gu, Y. Sun, and W. Wang, "Learning-based efficient graph similarity computation via multi-scale convolutional set matching," in *AAAI*, 2020.

[26] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," *ICML*, 2019.

[27] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *CCS*, 2017.

[28] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in neural information processing systems*, 1994, pp. 737–744.

[29] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*. Springer, 2016, pp. 850–865.

[30] R. R. Varior, M. Haloi, and G. Wang, "Gated siamese convolutional neural network architecture for human re-identification," in *European conference on computer vision*. Springer, 2016, pp. 791–808.

[31] J. Shin Yoon, F. Rameau, J. Kim, S. Lee, S. Shin, and I. So Kweon, "Pixel-level matching for video object segmentation using convolutional neural networks," in *IEEE ICCV*, 2017, pp. 2167–2176.

[32] X. Lu, W. Wang, C. Ma, J. Shen, L. Shao, and F. Porikli, "See more, know more: Unsupervised video object segmentation with co-attention siamese networks," in *CVPR*, 2019, pp. 3623–3632.

[33] H. He, K. Gimpel, and J. Lin, "Multi-perspective sentence similarity modeling with convolutional neural networks," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1576–1586.

[34] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[37] O. Melamud, J. Goldberger, and I. Dagan, "context2vec: Learning generic context embedding with bidirectional LSTM," in *Proceedings of the 20th SIGNLL conference on computational natural language learning*, 2016, pp. 51–61.

[38] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *KDD*, 2019.

[39] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *IEEE/ACM 40th ICSE*. IEEE, 2018, pp. 933–944.

[40] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in neural information processing systems*, 2013, pp. 926–934.

[41] S. H. Ding, B. C. Fung, and P. Charland, "Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization," in *IEEE S&P*, 2019.

[42] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[43] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and applications*, vol. 13, no. 1, pp. 113–129, 2010.

[44] K. Riesen, "Structural pattern recognition with graph edit distance," in *Advances in computer vision and pattern recognition*. Springer, 2015.

[45] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[46] K. Riesen, S. Emmenegger, and H. Bunke, "A novel software toolkit for graph edit distance computation," in *International Workshop on Graph-Based Representations in Pattern Recognition*. Springer, 2013.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *NIPS*, 2019, pp. 8026–8037.

[48] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[50] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, 1997.

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

[52] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, 1938.

[53] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

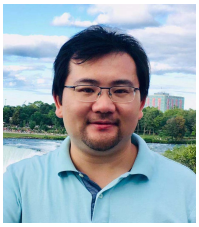
[54] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *ICLR*, 2016.

[55] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997.

[56] J. J. McGregor, "Backtrack search algorithms and the maximal common subgraph problem," *Software: Practice and Experience*, vol. 12, no. 1, pp. 23–34, 1982.

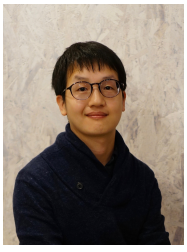
[57] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.
[58] D. B. Blumenthal and J. Gamper, "On the exact computation of the graph edit distance," *Pattern Recognition Letters*, 2018.
[59] L. Wu, I. E.-H. Yen, Z. Zhang, K. Xu, L. Zhao, X. Peng, Y. Xia, and C. Aggarwal, "Scalable global alignment graph kernel using random features: From node embedding to graph embedding," in *KDD*, 2019.
[60] T. Yoshida, I. Takeuchi, and M. Karasuyama, "Learning interpretable metric between graphs: Convex formulation and computation with graph mining," in *KDD*. ACM, 2019.

Xiang Ling received the B.E. degree from the China University of Petroleum (East China) in 2015. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University. His research mainly focus on the data-driven security, AI security and graph neural network.



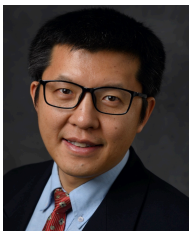
Lingfei Wu earned his Ph.D. degree in computer science from the College of William and Mary in 2016. He is a research staff member at IBM Research and is leading a research team (10+ RSMs) for developing novel Graph Neural Networks for various tasks, which leads to multiple IBM Awards including Outstanding Technical Achievement Award. He has published more than 70 top-ranked conference and journal papers and is a co-inventor of more than 30 filed US patents. He was the recipients of the Best Paper Award and Best Student Paper Award of several conferences such as IEEE ICC'19, AAAI workshop on DLGMA'20 and KDD workshop on DLG'19. His research has been featured in numerous media outlets, including NatureNews, YahooNews, Venturebeat, and TechTalks. He has co-organized 10+ conferences (AAAI, IEEE BigData) and is the founding co-chair for Workshops of Deep Learning on Graphs (with AAAI'21, AAAI'20, KDD'20, KDD'19, and IEEE BigData'19). He has served as Associate Editor for TNNLS, TKDD and IJIS.

Saizhuo Wang received his B.E. degree in Computer Science and Technology from Zhejiang University at Hangzhou, China in 2020. He is now pursuing his Ph.D. degree in Computer Science and Engineering at Hong Kong University of Science and Technology. His research mainly focuses on natural language processing and deep learning.



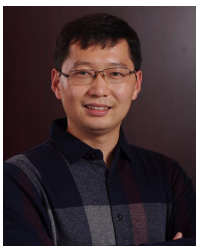
Tengfei Ma is a research staff member of IBM Research AI. Prior to joining IBM T. J. Watson Research Center, he obtained his Ph.D. from the University of Tokyo and worked in IBM Research Tokyo for one year. Before that he got his master's degree from Peking University and his bachelor's degree from Tsinghua University. His research interests have spanned a number of different topics in machine learning and natural language processing. Particularly, his recent research is focused on graph neural networks and their applications.

Fangli Xu earned her master degree in computer science from the College of William and Mary in 2018. She is a research engineer at Squirrel AI Learning. Her research interests mainly focus on machine learning, deep learning, and natural language processing, with a particular focus on AI + Education.



Alex X. Liu (Fellow, IEEE) received his Ph.D. degree in Computer Science from The University of Texas at Austin in 2006, and is currently the Chief Scientist of the Ant Group, China. Before that, he was a Professor of the Department of Computer Science and Engineering at Michigan State University. He received the IEEE & IFIP William C. Carter Award in 2004, a National Science Foundation CAREER award in 2009, the Michigan State University Withrow Distinguished Scholar (Junior) Award in 2011, and the Michigan State University Withrow Distinguished Scholar (Senior) Award in 2019. He has served as an Editor for IEEE/ACM Transactions on Networking, and he is currently an Associate Editor for IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Mobile Computing, and an Area Editor for Computer Communications. He has served as the TPC Co-Chair for ICNP 2014 and IFIP Networking 2019. He received Best Paper Awards from SECON-2018, ICNP-2012, SRDS-2012, and LISA-2010. His research interests focus on networking, security, and privacy. He is an IEEE Fellow and an ACM Distinguished Scientist.

Chunming Wu received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 1995. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. His research interests include software defined networks, proactive network defense, network virtualization, and intelligent networks.



Shouling Ji is a ZJU 100-Young Professor in the College of Computer Science and Technology at Zhejiang University and a Research Faculty in the School of Electrical and Computer Engineering at Georgia Institute of Technology (Georgia Tech). He received a Ph.D. degree in Electrical and Computer Engineering from Georgia Institute of Technology, a Ph.D. degree in Computer Science from Georgia State University, and B.S. (with Honors) and M.S. degrees both in Computer Science from Heilongjiang University. His current research interests include Data-driven Security and Privacy, AI Security and Big Data Analytics. He is a member of ACM, IEEE, and CCF and was the Membership Chair of the IEEE Student Branch at Georgia State University (2012-2013). He was a Research Intern at the IBM T. J. Watson Research Center. Shouling is the recipient of the 2012 Chinese Government Award for Outstanding Self-Financed Students Abroad.

APPENDIX

A. More Details about Datasets

For **FFmpeg**, we prepare the CFGs as the benchmark dataset to detect binary function similarity. First, we compile *FFmpeg 4.1.4* using 2 different compilers (*i.e.*, *gcc 5.4.0* and *clang 3.8.0*) and 4 different compiler optimization levels (*O0-O3*), which produces a total of 8 different types of compiled binary files. Second, these 8 generated binaries are disassembled with 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 [27] 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 as follows: *# of string constants*, *# of numeric constants*, *# of total instructions*, *# of transfer instructions*, *# of call instructions*, and *# of arithmetic instructions*.

B. Detailed Experimental Settings for Baseline Models

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

SimGNN: SimGNN firstly adopts a 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 the neural tensor network (NTN) to interact two graph-level embedding vectors 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 task, the output dimensions for the three-layer GCN 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 loss function of mean square error 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.

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

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

To fairly compare our models with SimGNN in evaluating the graph-graph classification task, we adjust the settings of SimGNN as follows. We follow the same architecture of SimGNN in the graph-graph regression task 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 task. As for training, we use the cross-entropy loss function to train the model and set the number of epochs to 100. Other training hyper-parameters are kept the same as the graph-graph regression task.

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 computes a graph-level embedding vector for each graph.

For the graph-graph classification task, we use a 1-layer MLP as the node/edge encoder and set the number of rounds of propagation to 5. The dimension of the node embedding is set to 32, and the dimension of graph-level representation vectors 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 a learning rate of 0.001 and a batch size of 10.

To enable fair comparisons with GMN for the graph-graph regression task, we adjust the GMN 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 the mean square loss function with batch size 128. Other settings remain the same as the graph-graph classification task.

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 a pair of input graphs, then turns the two sets of node embeddings into a similarity matrix consisting of the pairwise node-node interactions, feeds these matrices 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 task, 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: $Conv(6, 1, 1, 16)$, $Max(2)$, $Conv(6, 1, 16, 32)$, $Max(2)$, $Conv(5, 1, 32, 64)$, $Max(2)$, $Conv(5, 1, 64, 128)$, $Max(3)$, $Conv(5, 1, 128, 128)$, $Max(3)$. Numbers in $Conv()$ represent the window size, kernel stride, input channels, and output channels of the CNN layer, respectively. The number in $Max()$ denotes the pooling size of the max pooling operation. 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

TABLE IX
THE GRAPH-GRAPH REGRESSION RESULTS OF NGMN MODELS WITH DIFFERENT NUMBERS OF PERSPECTIVES IN TERMS OF mse, ρ, τ, $p@10$ & $p@20$.

Datasets	Model	mse (10^{-3})	ρ	τ	$p@10$	$p@20$
AIDS700	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
LINUX1000	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 X
THE GRAPH-GRAPH REGRESSION RESULTS OF NGMN MODELS WITH DIFFERENT GNNS IN TERMS OF mse, ρ, τ, $p@10$ & $p@20$.

Datasets	Model	mse (10^{-3})	ρ	τ	$p@10$	$p@20$
AIDS700	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
LINUX1000	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

TABLE XI
THE GRAPH-GRAPH REGRESSION RESULTS OF NGMN MODELS WITH DIFFERENT NUMBERS OF GCN LAYERS IN TERMS OF mse, ρ, τ, $p@10$ & $p@20$.

Datasets	Model	mse (10^{-3})	ρ	τ	$p@10$	$p@20$
AIDS700	NGMN-(1 layer)	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
LINUX1000	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

for training, the loss function of mean square error 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 models with GraphSim in our evaluation, we also adjust GraphSim to solve the graph-graph classification task. We follow the same architecture of GraphSim in the graph-graph regression task except that seven connected layers are used instead of eight. The output dimension of the final connected layers is set to 2, and we apply a Softmax operation over it to get the predicted binary label for the graph-graph classification task. As for training, we use the cross-entropy loss function to train our models and set the number of epochs to 100. Other training hyper-parameters are kept the same as the graph-graph regression task.