DeepGraphlet: Estimating Local Graphlet Frequencies for Billion-scale Graphs

Anonymous Author(s)

ABSTRACT

Local graphlet frequencies (LGF) indicate the distribution of graphlets (i.e., small connected subgraph patterns) adjacent to each node in the network. Accordingly, it is emerging as a powerful tool for characterizing the local topology structures of networks and has been widely applied in various domains, ranging from biology to network science. However, the counting of local graphlets is associated with prohibitive computational costs in the context of large real-world graphs, and thereby represents a long-standing research problem. Accordingly, in this paper, we make the first attempt to compute LGF for *billion-scale* graphs by transforming the task into a machine learning problem. To achieve this, we propose a multi-layer graph neural network (GNN) -based framework: DeepGraphlet. In more detail, we propose the novel *k-tuple features* and theoretically prove that GNNs with it can exceed the bound of the expressiveness for capturing graph structural information. Moreover, DeepGraphlet utilizes both the cross- and inner-order relationships among graphlets. Furthermore, a multitask mechanism is proposed to utilize the graphlet relationships better and to accelerate the computation by learning different-order LGFs simultaneously in a unified framework. When used together, these strategies guarantee the scalability of DeepGraphlet. To empirically validate the proposed model, we conduct experiments on nine graphs. Experimental results show that our approach is not only able to improve the estimation accuracy compared with other approximate algorithms (+80% on average), but also achieves significant speedup (e.g., 749x speedup on a hundreds of millions-scale graph). Furthermore, the experimental results illustrate DeepGraphlet's ability to handle billion-scale graphs. Our code is available at https://github.com/deepgraphlet/DeepGraphlet.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

local graphlet frequencies, billion-scale graph

ACM Reference Format:

Anonymous Author(s). 2022. DeepGraphlet: Estimating Local Graphlet Frequencies for Billion-scale Graphs. In *KDD '22: SIGKDD CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, August 14-18, Washington DC Convention Center.*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

KDD '22, August 14-18, 2022, Washington DC Convention Center

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Local graphlets of a given network refer to small induced subgraph patterns adjacent to each node (see Figure 1 for examples). Unlike some global network properties, such as degree distribution, the frequencies of local graphlets (i.e., the normalized number of each graphlet associated with a particular node) provide an insightful characterization of both individual nodes and the topological structures of networks. Therefore, local graphlet frequencies (LGF) has been widely applied in various domains, including network science [3, 22, 43], biological science [23], anomaly detection [45], and social networks [37]. For example, LGF can be used to compute general graph features, such as local clustering coefficients [46], and analyze the collective dynamics of small-world networks [39]. However, counting local graphlets is a computationally intensive task. The time complexity of the straightforward method of enumerating a k-order graphlet is $O(|V|^k)$, where $|V|$ denotes the number of nodes. Nowadays, in real-world applications, graphs can reach hundreds of millions of nodes and billions of edges, or even more. Thus, the time complexity of processing such large graphs is often unfeasibly enormous.

Due to its importance, graphlet counting has attracted a significant amount of research interest, including both exact and approximate algorithms. For exact algorithms, some works compute k-order graphlets by enumerating all subgraphs of the same order [24, 41]. In an attempt to accelerate the computation, some studies have analyzed the relations between different graphlets [1, 15, 28]. However, the problem of high time complexity on large graphs remains. As for approximate algorithms, most existing studies in this area address global graphlet counting [30], the occurrence of graphlets in the whole graph, while only few works deal with local graphlets [2, 10]. Still, these methods suffer from their high time complexity, which has a power relation with the order of local graphlets. Therefore, these methods is difficult to scale to large graphs for high-order graphlets. In practice, most existing algorithms can only deal with 3- or 4-order graphlets for million-scale graphs.

Accordingly, in this paper, we focus on approximating node-centric LGF for billion-scale graphs. To achieve this, we propose to turn the LGF computing problem into a learning problem. Generally speaking, our idea is to use an inductive graph neural network to map the node representations into LGF. In the training phase, we minimize the difference between the LGF predicted by our model and the ground truth calculated by the exact algorithm on small graphs. When it comes to inference, we use the trained model directly on new large-scale graphs to compute the LGF.

The problem of computing LGF for large graphs via GNN is non-trivial and associated with many challenges. First, LGF is highly

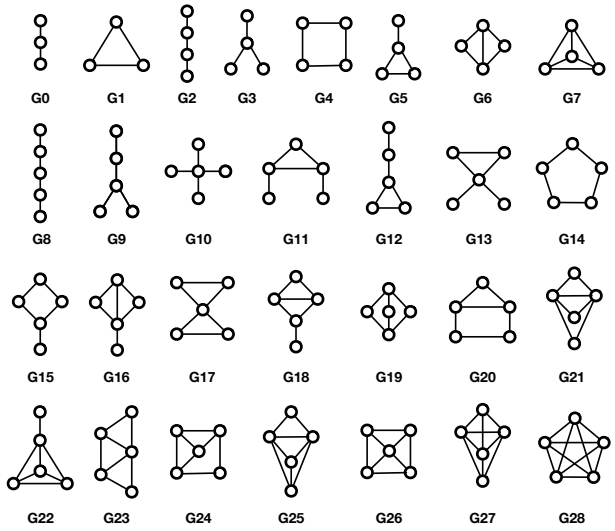

Figure 1: Illustration of 3-, 4- and 5-order graphlets.

relevant to the graph structural information. However, as discussed in many existing works, the expressive power of GNNs for graph structural information is limited [11, 42]. Thus, the question of how to exceed the limitation of GNNs and better capture the local structural information around nodes remains a crucial challenge for us. Secondly, the exact counting algorithm has shown that the relationships between different graphlets can inherently make the counting process more efficient [1]. Therefore, making deep neural models fully utilize these graphlets relationships carries abreast challenges and opportunities to us. Thirdly, as mentioned above, counting local graphlets is a computationally intensive task. Similar works that adopt graph neural networks for graphlet counting [11, 35] are unable to handle large graphs because their time complexity is super-linear to the number of nodes. We aim to accelerate the computation and enable the GNN-based model to handle billion-scale graphs.

To address the above challenges, we propose a multi-layer GNN-based framework, named DeepGraphlet, with three novel mechanisms. First, we propose a novel *k-tuple feature* for each node as the initial identifier, which equips DeepGraphlet with more graph structural information. We theoretically prove that GNNs with k-tuple features exceed the bound of general GNNs' expressive power in the graph isomorphism problem — the 1-order Weisfeiler-Lehman graph isomorphism test (1-WL). We propose an efficient algorithm for extracting the k-tuple features with the time complexity linear in $|V|+|E|$. Secondly, in order to capture and utilize cross- and inner-order relationships among graphlets, DeepGraphlet incorporates GNNs with a *hierarchical structure*. Thirdly, the GNNs are efficiently trained by a *multi-task mechanism*, which enables different-order LGFs can be inferred simultaneously in a unified framework and thus accelerates the computation. Another thing worth noting is that, given the properties of GNNs, the model's parameters are independent of the graph scale; hence, the proposed model is naturally inductive.

We conduct experiments on nine real-world graphs with numbers of edges ranging from millions to billions. The results demonstrate that DeepGraphlet achieves the best results when it comes to the computation of different-order LGFs in nearly all settings. In terms of effectiveness, compared with several baselines, DeepGraphlet can significantly improve the estimation accuracy (+80% on average). As for efficiency, DeepGraphlet can achieve significant speedup compared to the baseline (e.g., 749x speedup on a hundreds of millions-scale graph). We further compute LGF on graphs with billions of edges, achieving 59x speedup compared to sampling algorithms. It is worthwhile to highlight our contributions as

1) We turn the node-centric LGF computing task into a learning problem, such that an inductive model can be applied to estimating different-order LGFs on billion-scale graphs.
2) We propose a novel framework, DeepGraphlet, which is equipped with three novel mechanisms to guarantee its estimation performance and scalability. We theoretically prove that our model, with the help of the proposed k-tuple features, can exceed the bound of the expressive power of general GNNs.
3) We conduct experiments on nine real-world graphs with numbers of edges ranging from millions to billions. The results prove that DeepGraphlet has high effectiveness and efficiency.

2 RELATED WORKS

2.1 Graphlet Counting Algorithms

Exact graphlet counting. Exact graphlet counting is a computationally intensive task. To solve the challenge of computational time complexity, many works have been developed to accelerate the computation. Some works focus only on 3-order graphlet counting [6, 19, 34, 36], while others aim at achieving higher-order graphlet computing. MFINDER [24] and FANMOD [41] are enumeration-based backtracking algorithms. ORCA [15] analyzes the relationships among different graphlets for accelerating computation. PGD [1] proposes an efficient algorithm that can compute 3- and 4-order graphlet counts and scale to large graphs with millions of nodes. Escape [28] allows for the computation of 3-, 4- and 5-order global graphlet counts on tens of millions of edges. However, as discussed above, the problem of high time complexity on large graphs remains.

Approximate graphlet counting. Approximate algorithms are developed to accelerate graphlet counting further. Most of these works estimate global graphlets [26]. Wernicke [40] uses random enumeration developed based on exact enumeration. GUISE [7] uses random walk and MCMC sampling method, while Motivo [9] uses color coding and adaptive sampling to count graphlets faster. Compared with works focused on global graphlet counting, relatively few works aim at local graphlet counting due to its difficulty. Ahmed [2] approximates 3- and 4-order edge-centric local graphlet counts, while Chen [10] can approximate 3- and 4-order node-centric local graphlet counts. To the best of our knowledge, no existing local graphlet counting approximate algorithms can deal with 5-order or higher-order node-centric local graphlets counting in large graphs. Some global graphlet counting algorithms can be

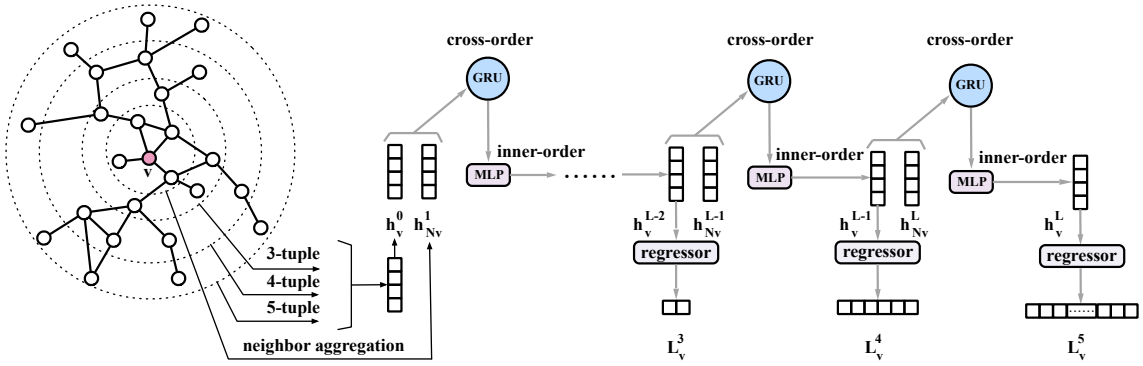

Figure 2: The structure of the DeepGraphlet framework. An example of computing up to 5-order LGF.

transformed into local graphlet counting. However, to accurately estimate local graphlets around each graph node, these algorithms require much more sampling times than global estimation. Thus, the transformed global graphlet counting algorithm is not feasible. In summary, efficient higher-order graphlet counting remains a challenge for us.

2.2 Graph Neural Networks

In recent years, GNNs has attracted substantial research interests. First introduced in [33], GNNs have subsequently demonstrated their ability to capture graph structural information. Recent years, Graph Convolutional Networks (GCN) [12, 16] have achieved promising results on several tasks. Related works include GAT [38], which incorporates an attention mechanism, and GraphSage [14], which can scale to large graphs via neighborhood sampling. [42] analyzed the relationship between GNNs and the 1-order Weisfeiler-Lehman graph isomorphism test (1-WL) and proposed a powerful model GIN.

Applications. GNN achieves great performance on various tasks, the most common of which is node classification. [13, 21] approximated betweenness centrality using GNN, while [4, 5] used GNN to compute graph similarity in an end-to-end framework. Recently, some works have also utilized GNNs to conduct substructure counting. Liu [20] learned a neural model to count subgraph isomorphisms, while Ying [29] found frequent subgraphs in a large target graph via a GNN encoder and motif search procedure. [11, 35] proposed to improve GNNs' graphlet counting ability by employing higher-order GNNs. However, these approaches cannot scale to million-scale graphs due to their high time complexity.

3 PROBLEM DEFINITION

Consider a graph $G = (V, E)$, where V denotes its node set and E represents its edge set. The local graphlet frequency L_u^{kj} of a k-order graphlet Gj for node u is defined as the probability that a random k-order subgraph in G with u included can induce the same structure of Gj. Formally, we can define L_u^{kj} as follows:

$$L_u^{kj} = \frac{N_u^{kj}}{\sum_{j'=1}^{m} N_u^{kj'}} \tag{1}$$

where N_u^{kj} is the number of k-order graphlet Gj involving node u, m is the number of different types of k order graphlets. Intuitively, L_u^{kj} represents the percentage of the k-order graphlet Gj among all k-order graphlets that contain node u. Figure 1 enumerates all 3-order (G0–G1), 4-order (G2–G7), and 5-order (G8–G28) graphlets. Consider G3 as an example; its frequency for node u is the ratio between the number of G3 that contain u and the number of all 4-order graphlets (G2–G7) that include u.

Problem. The goal of this work is *to estimate the local graphlet frequency distribution* $\mathbf{L}_u^k = \{L_u^{kj}\}$ *for each node* u *in* V. For example, a node's 5-order graphlet frequency distribution is a 21-length vector, where each element corresponds to the percentage of one graphlet from G8 to G28.

Traditionally, most related works have focused on the exact count of these graphlets, either locally or globally; examples include the Escape [28] and PGD [1] algorithms. However, the time complexity of these methods is very high (usually super-linear in $|V| + |E|$), and to date, they can only handle graphs with hundreds of millions of edges. Prior attempts to approximate the graphlet counts/distributions focused primarily on global estimations. In contrast, local estimation is a more challengeable task [30]. We instead focus on estimating the local graphlet frequency for billion-scale graphs in this work.

4 OUR APPROACH

Overview. To estimate the local graphlet frequency (LGF) for a graph, traditional methods typically leverage graph theory to develop approximate algorithms capable of directly enumerating the graphlets of interest [30]. In this work, we propose to leverage a machine learning pipeline for this task. In general terms, we take small graphs, compute their LGFs using specific exact algorithms, and then train a model to fit the computed LGFs. The promise is that the model can learn the correlation between the structural information and LGF.

To this end, we present the DeepGraphlet framework in this section. Figure 2 presents an overview of our approach. Given a particular graph, the first step of DeepGraphlet is to extract the novel *k-tuple features*, which reflects the local structural information of each node. These features are then fed to a multi-layer GNN

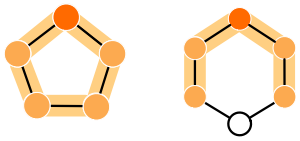

Figure 3: Illustration of k-tuple features' expressive power.

to quantify the correlation between a node's local structural information and its LGF. Different layers of DeepGraphlet then output the computed LGFs with different orders.

In more detail, we aim to solve the following three major technical issues in the rest of this section: 1) LGF is highly relevant to the structure of the given graph. However, as many existing works have mentioned, the expressive power of GNNs for graph structural information is limited [11, 42]. Efficiently and effectively resolving this issue thus becomes the basis of our approach. 2) LGFs with different orders are correlated with each other. Ignoring such dependency and computing different-order graphlets independently can be inefficient and inaccurate. Thus, answering how to model the dependency among graphlets is vital to model design. 3) As we discussed above, a significant challenge encountered in LGF computation is that of how to scale to large graphs. Therefore, in each part of the model, how to promote the model's efficiency and enable it to handle billion-scale graphs remains a crucial problem we aim to answer.

4.1 Extracting Structural Information

Given that our goal is to handle graph data, we naturally choose GNN as the underlying model of our framework. Moreover, inspired by traditional graphlet counting algorithms like PGD [1], which is conceptualized as counting a node's graphlets by gathering its neighborhoods' information, we choose GNN because it also follows a computational paradigm of neighborhood aggregation. Furthermore, the inductive property of GNN makes it practical for training the model on small graphs and generalizing the trained model to large-scale graphs.

However, the expressive power of GNNs is limited for graph structural information. Notably, computing LGF can be regarded as a particular form of the graph isomorphism problem. However, GNN's ability to test graph isomorphism has been proven to be at most as powerful as the 1-order Weisfeiler-Lehman (1-WL) graph isomorphism test [42]. Figure 3 presents an example, in which all nodes in the graphs have two neighbors. For standard GNNs with the same node feature initialization, all nodes always have the same representation in each GNN layer. Accordingly, although they have different LGFs (e.g., the left graph has a circle made up of five nodes, while the five nodes in the right graph can only form a simple path), traditional GNNs find it challenging to distinguish them.

K-tuple features. To better capture the graph structural information and help GNNs to exceed the limitation of the 1-WL test, we propose the novel *k-tuple features*. The k-tuple is a tuple containing k nodes, and the induced subgraph of it is connected. We heuristically sample a fixed number of k-tuples for each node, and regard the appearance times of the isomorphism types of k-tuples' induced subgraph as the feature value.

In more detail, we design an efficient heuristic algorithm to extract k-tuple features. The pseudo-code is in Appendix B. For a particular node v, we maintain a node list initialized with v itself. We then perform $k-1$ sampling steps. In each step, we first randomly select a node s from the node list with the probability proportional to the degree of each node. In the next step, we randomly select a neighbor of s with probability proportional to the neighbor's degree via the alias algorithm and add the neighbor to the node list. After we have collected k nodes, we identify the isomorphism type of the graph induced by these nodes. We repeat the above process a fixed times and regard the number of times each isomorphism type appears as the feature value. It is worth noting that the sampling algorithm's time complexity can be described linearly in $|V| + |E|$, which ensures the efficiency of our model.

Theoretical analysis. With the generated k-tuple features, our model can exceed the bound of the expressive power of GNNs. For the example in Figure 3, 1-WL cannot distinguish the nodes in the two graphs. With the 5-tuple feature, however, the GNNs can distinguish these nodes at initialization, as the corresponding values in the k-tuple feature of 5-circle and 5-path are different. More formally, we have Theorem 4.1 (the proof for which is provided in Appendix A.1).

Theorem 4.1. *GNNs with k-tuple features can exceed the expressive power of 1-WL in the graph isomorphism problem.*

Thus, GNNs with the k-tuple features, have more powerful structural identification ability. Compared with other features that improve the identification ability of GNNs, k-tuple features have several advantages. Specifically, compared to one-hot and random feature [32], k-tuple features are inductive and structure related, which benefits the final tasks; compared to features like distance encoding [18], k-tuple features are efficient as the time complexity of extraction is linearly in $|V| + |E|$.

Furthermore, we analyze the relationship between k-tuple features and the k-WL in Theorem 4.2 (the proof of which is in Appendix A.2).

Theorem 4.2. *The (k-1)-WL algorithm is not stronger than GNNs with k-tuple features in the graph isomorphism problem where* $3 \leq k \leq 4$.

K-WL algorithm is a more powerful version of the 1-WL algorithm designed for the graph isomorphism problem. The concept of k-tuple exists in both algorithms. The k-tuple features are extracted as the node feature initialization for the further GNN layers. Meanwhile, k-WL constructs a new graph whose nodes are k-tuples of the original graph's nodes. The neighbor aggregation is then conducted on the newly constructed graphs. The expressive power of GNNs with k-tuple features and k-WL is analyzed in the above theorem. For example, GNNs with 4-tuple features, can correctly identify the isomorphism of two graphs, while 3-WL fails.

4.2 Capturing Graphlet Relationship

To further promote the effectiveness of our model, in this section, we propose to capture the relationships among graphlets in our model. Naturally, relationships exist among graphlets. For example, as illustrated in Figure 4, 4-order graphlets can be obtained by adding

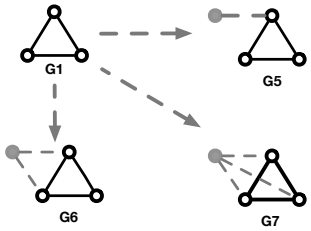

Figure 4: An example of relationships between lower-order and higher-order graphlets.

one more node to 3-order graphlets. Thus, by using the information of 3-order graphlets, we can count the 4-order graphlets more easily. Properly utilizing such relationships can accelerate the graphlet counting process, which is also the motivation behind PGD [1].

DeepGraphlet is designed to capture the cross- and inner-order relationships in order to compute LGF more accurately and efficiently with powerful components and a novel multi-task mechanism.

Cross-order relationship. To better capture the graph structure information, we aim to utilize the cross-order relationships. In the GNN propagation process, the previous layers' node representations contain the lower-order information, while the aggregated neighbor representation at this layer is a form of higher-order information. Simply add the information of the previous layer, and the newly aggregated information, as in normal GNNs, cannot extract the cross-order relationships well. Thus, to enable the model to learn more flexible and complex cross-order relationships, we propose to adopt Gated Recurrent Unit (GRU):

$$h^i_{N(v)} = \sum_{u \in N(v)} \frac{1}{d_v} h^{i-1}_u \tag{2}$$

$$z^i_v = \sigma(W^i_z h^i_{N(v)} + U^i_z h^{i-1}_v) \tag{3}$$

$$r^i_v = \sigma(W^i_r h^i_{N(v)} + U^i_r h^{i-1}_v) \tag{4}$$

$$\widetilde{h}^i_v = tanh(W^i h^i_{N(v)} + U^i(r^i_v \odot h^{i-1}_v)) \tag{5}$$

$$\hat{h}^i_v = (1 - z^i_v) \odot h^{i-1}_v + z^i_v \odot \widetilde{h}^i_v \tag{6}$$

where h^{i-1}_u is the previous layer's embedding of node u, $N(v)$ is the neighbors of node v, d_v is the degree of node v, and $\odot$ is an element-wise dot operation. Moreover, we use GRUs of different parameters in each propagation step, as graphlets of different orders have different cross-order relationships. Note that, in DeepGraphlet, we choose the mean aggregate function (aggregator) to define how GNN aggregates the information of neighbors. The reason is that the mean aggregator can learn the distribution well [42], while LGF is a distribution of graphlets.

Inner-order relationship. As analyzed in [1], relationships also exist within graphlets of the same order; accounting for these is essential for graphlet counting. Thus, after we capture the cross-order relationships and generate the combined higher-order information, we use a two-layer multi-layer perceptron (MLP) to capture the inner-order relationships. We then output the higher-order node representation. The non-linear transformation allows the model to capture complex inner-order relationships.

$$h^i_v = ReLU(W^{i2} ReLU(W^{i1} \hat{h}^i_v + b^{i1}) + b^{i2}) \tag{7}$$

Multitask mechanism. We design a novel multitask mechanism to further utilize the graphlet relationships by explicitly adding the supervise signals of different-order LGFs. In addition to the graphlet relationships, there is another factor that inspires us: in the computing process, we only require the information of up to (k-1) hop neighbors to compute the k-order LGF of node v. Thus, the computation of lower-order graphlets naturally requires smaller GNN propagation times. Based on these inspirations, we propose a novel multitask mechanism that enables us to compute different-order LGFs simultaneously.

More specifically, in an L-layer DeepGraphlet, to output the k-order LGF, we insert an output layer after the $L-(K-k)^{th}$ GNN layer; here K is the maximum order we want to compute, and k ranges from 3 to K. The output layer after the l^{th} GNN layer is designed as follows:

$$\hat{\mathbf{L}}^l_u = Softmax(ReLU(W^{o2} ReLU(W^{o1} \hat{h}^l_u + b^{o1}) + b^{o2})) \tag{8}$$

We pass the node representation through a two-layer MLP, then employ a softmax function to map the output to LGF.

To compute up-to K-order LGF (ranging from 3-order to K-order), we add different output layers after the $(L-(K-3))^{th}$, $(L-(K-4))^{th}$, ..., L^{th} GNN layers. We compute the final loss by adding the losses of different-order LGFs, then conduct the back propagation. As LGF represents the distribution of the graphlets, we use Kullback-Leibler (KL) divergence to evaluate the loss between the predicted LGF and the real LGF. For the k-order LGF:

$$loss_k = \frac{1}{|V|} \sum_{u \in V} \sum_{i=1}^{m} \hat{L}^{ki}_u log \frac{\hat{L}^{ki}_u}{L^{ki}_u} \tag{9}$$

where m is the number of different k-order graphlets.

Being equipped with the multi-task mechanism, DeepGraphlet can run substantially faster than standard GNNs, since DeepGraphlet can compute 3 to K-order LGF in a single run. In the meantime, for standard GNNs, we need to train different models to compute different-order graphlets. In addition to the increase in efficiency, training the model with a multi-task mechanism also boosts the model's performance in two ways: 1) hard parameter sharing multi-task learning enables different tasks to share parts of the neural network layers, which causes the model to attain better generalization ability than when tasks are trained separately; 2) as we discussed above, lower-order graphlets can assist in the computation of high-order graphlets. By explicitly enforcing different GNN layers to learn different-order LGFs, DeepGraphlet can compute high-order LGF more accurately.

4.3 Complexity Analysis

The time complexity of DeepGraphlet contains two parts: k-tuple feature generation and GNN runtime.

K-tuple feature generation. The k-tuple feature generation time complexity is $O(|E| + n \cdot |V| \cdot |G_k| \cdot k!)$; here n is the sampled k-tuple number for each node, $|V|$ is the number of nodes, $|G_k|$ is the number of k-order graphlets, and $k!$ is the k factorial. We use the

Dataset	$\|V\|$	$\|E\|$	avg. degree
Slashdot	82,168	504,230	6.1
Artist	50,515	819,090	32.4
Google	875,713	4,322,051	4.9
Topcats	748,766	5,522,409	7.4
BerkStan	685,230	7,600,595	11.1
Patents	3,774,768	16,518,947	4.4
LJ	3,997,962	34,681,189	8.7
Orkut	3,072,441	117,185,083	76.3
Friendster	65,608,366	1,806,067,135	27.5

Table 1: Overview of Datasets

alias algorithm with pre-processing time complexity $O(|E|)$ and then sample the nodes' neighbors with a probability proportional to their degrees in $O(1)$. Moreover, $n \cdot |V|$ is the total number of sampled k-tuples, while $|G_k| \cdot k!$ denotes the time required to identify the isomorphism type of the induced subgraph of k-tuples. When k is small, the isomorphism identification algorithm can be optimized to k^2 using hashing.

GNN complexity. The single execution time of GNN is linearly proportional to $|E|$, $O(T \cdot |E| \cdot h)$; here T is the number of GNN layers, $|E|$ is the number of edges, and h is the dimension of the neural network's hidden size.

Overall, the time complexity of DeepGraphlet has a linear relationship with the number of nodes and edges in a graph.

5 EXPERIMENTS

5.1 Experimental Setup

In our experiments, we focus on computing 3-, 4- and 5-order local graphlet frequencies, considering that higher-order LGF are rarely used in practical applications.

Datasets. We adopt nine graph datasets of different scales to validate the effectiveness and efficiency of our model. In more detail, Slashdot [17], Artist [31], LJ, Orkut and Friendster [44] are social networks, BerkStan [17], Google [17] and Topcats [47] are web graphs, and Patents [47] is a citation network. Detailed statistics of each dataset are presented in Table 1. The training and validation graphs are generated by sampling connected subgraphs from a real graph. We repeat the sampling steps to obtain several small graphs for training and validation. For each graph, we randomly sample 15 small connected subgraphs containing $\frac{|V|}{30}$ nodes; here, 10 sampled graphs are used for training, while the remaining 5 are validation graphs. After training the models on the sampled small graphs, we test the performance on the original real graphs.

Baselines. We compare our model with two categories of baselines. The first category is the approximate graphlet counting algorithm. To the best of our knowledge, there is no sampling-based local graphlet counting algorithm capable of computing up to 5-order LGF. Thus, we included transformed global graphlet counting algorithm and neural network-based models as our baselines. Motivo [9] is the state-of-the-art global graphlet counting algorithm based on color coding. We transform it into the local version as a baseline. For neural network-based baselines, we consider GCN [16], GIN [42], rGIN [32] and MLP+K. GIN is an expressive model as powerful as 1-WL Test in the graph isomorphism problem. rGIN further improves the expressive power of GIN by incorporating random features. We further consider several variants of our model as baselines; these are referred to as DeepGraphlet-K, DeepGraphlet-R and DeepGraphlet-M, and these variants are implemented by removing the k-tuple features, the cross-order relation extraction mechanism, and the multi-task mechanism respectively from DeepGraphlet. It should however be noted that although deep models like LRP [11], k-GNNs [25], RNP-GNNs [35] and distance encoding [18] are powerful in the graph isomorphism problem, we do not regard them as baselines, since their high time complexity makes them unaffordable even on our smallest graph. For exact graphlet counting algorithms, we adopt Evoke [27], the state-of-the-art exact counting algorithm for local graphlets. To the best of our knowledge, Evoke is the most efficient method that is capable of handling up-to 5-order LGF computation.

Evaluation metrics. We compare the performance of different models in terms of KL divergence between the estimated results and those computed by the Evoke. LGF computation is not only a regression problem. Furthermore, LGF is a distribution intrinsically. Thus, we choose KL divergence as it is a standard metric to measure the difference between distributions. We further compare the efficiency of models by reporting their inference time on test data.

We further provide implementation details in Appendix C.

5.2 Results of Effectiveness

We report the effectiveness of DeepGraphlet compared to baselines in Table 2. It should be noted that smaller KL divergence indicates a smaller distance between the predicted results and the ground truth. For overall performance, DeepGraphlet beats all baselines in nearly all datasets. More specifically, DeepGraphlet achieves a signficant loss reduction compared to transformed global graphlet counting algorithm motivo. When transformed into estimating local graphlet counting for each node in a graph, the global graphlet counting algorithm needs significantly larger sample times. We set the sample times of motivo as $|V| * 100$ for each task, which has already taken orders of magnitude running time than our models. For GIN, DeepGraphlet decreases by 81% in terms of KL divergence compared to GIN. This is because GIN uses a sum aggregator, and the numeric scale of the sum aggregator's output is dependent on degrees. Therefore, the average degree gap between the train and test graphs makes GIN's performance sub-optimal. As for the GCN and rGIN, our model yields KL divergence values that are 75% and 72% lower respectively. This is due to the ability of k-tuple features to capture structural information, which is essential to the LGF computation. Due to being equipped with k-tuple features, our model's expressive ability is stronger than the baselines.

For the ablation studies shown in Table 3, we illustrate the effectiveness of three mechanisms of DeepGraphlet. DeepGraphlet-K is the proposed model with the k-tuple features removed. DeepGraphlet exhibits a noticeable performance improvement compared to DeepGraphlet-K (76% KL divergence reduction), which demonstrates the importance of K-tuple features for LGF computation. The K-tuple features offer more structural information to the model, which enables it to compute the LGF more accurately. Moreover, compared to DeepGraphlet-R, DeepGraphlet exhibits an obvious performance improvement (46% KL divergence reduction). This is

Overall \ Data.	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
motivo	3.785±0.005	3.341±0.015	4.339±0.007	4.673±0.048	4.132±0.002	3.462±0.002	4.124±0.011	4.130±0.014
GCN	0.788±0.017	0.544±0.042	1.608±0.084	1.524±0.023	1.077±0.061	0.330±0.041	0.732±0.026	0.672±0.018
GIN	67.69±41.52	5.804±1.594	0.803±0.111	2.616±1.160	1.035±0.126	0.210±0.015	0.640±0.017	3.205±1.158
rGIN	2.069±0.522	1.419±0.663	0.649±0.108	1.041±0.127	0.793±0.084	0.166±0.011	0.610±0.026	1.316±0.564
DeepGraphlet	**0.251±0.030**	**0.151±0.011**	**0.273±0.013**	**0.292±0.009**	**0.386±0.015**	**0.061±0.002**	**0.181±0.008**	**0.161±0.005**
3 - LGF \ Data.	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
motivo	0.514±0.002	0.460±0.001	0.561±0.002	0.520±0.001	0.590±0.002	0.471±0.000	0.485±0.000	0.428±0.000
GCN	0.024±0.010	0.027±0.006	0.140±0.037	0.041±0.016	0.045±0.003	0.021±0.002	0.043±0.002	0.034±0.002
GIN	0.050±0.012	0.055±0.013	0.034±0.002	0.079±0.047	0.068±0.025	0.024±0.004	0.050±0.010	0.040±0.008
rGIN	0.093±0.039	0.046±0.006	0.023±0.009	0.046±0.009	0.042±0.011	0.015±0.000	0.035±0.004	0.035±0.003
DeepGraphlet	**0.010±0.006**	**0.004±0.001**	**0.006±0.001**	**0.004±0.001**	**0.005±0.001**	**0.004±0.000**	**0.002±0.000**	**0.001±0.000**
4 - LGF \ Data.	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
motivo	1.147±0.002	1.028±0.007	1.541±0.005	1.520±0.010	1.277±0.008	0.993±0.001	1.381±0.001	1.410±0.005
GCN	0.287±0.029	0.195±0.009	0.489±0.027	0.472±0.014	0.348±0.024	0.099±0.016	0.209±0.009	0.190±0.011
GIN	2.804±1.686	0.951±0.123	0.255±0.053	1.446±1.067	0.231±0.020	0.076±0.009	0.199±0.017	0.269±0.050
rGIN	0.704±0.132	0.315±0.011	0.162±0.028	0.305±0.091	0.230±0.014	0.056±0.006	0.178±0.016	0.601±0.291
DeepGraphlet	**0.094±0.013**	**0.045±0.004**	**0.057±0.003**	**0.069±0.004**	**0.091±0.005**	**0.018±0.001**	**0.037±0.003**	**0.036±0.002**
5 - LGF \ Data.	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
motivo	2.123±0.005	1.853±0.008	2.237±0.015	2.632±0.043	2.265±0.009	1.999±0.002	2.258±0.010	2.291±0.018
GCN	0.476±0.020	0.322±0.044	0.979±0.073	1.010±0.020	0.684±0.040	0.209±0.035	0.481±0.016	0.448±0.016
GIN	64.83±42.21	4.799±1.577	0.513±0.075	1.091±0.674	0.735±0.116	0.111±0.009	0.391±0.014	2.897±1.144
rGIN	1.273±0.501	1.058±0.667	0.464±0.096	0.690±0.196	0.521±0.071	0.095±0.008	0.398±0.011	0.681±0.283
DeepGraphlet	**0.147±0.016**	**0.102±0.009**	**0.210±0.011**	**0.219±0.005**	**0.290±0.012**	**0.039±0.001**	**0.142±0.005**	**0.123±0.003**

Table 2: The results of training on sampled sub-graphs of a graph and testing the model's performance (KL-divergence) on the same graph. The smaller value means better performance.

due to the fact that flexible functions can learn the complex relationships between graphlets, which is useful for LGF computation. As for the comparison with the variant DeepGraphlet-M, DeepGraphlet achieves 3%+ better overall performance, while running much faster as introduced later in Sec 5.4.

5.3 Results of Transfer Ability

In this section, we investigate the transfer ability of DeepGraphlet. Specifically, we train models on sampled small graphs of one graph. Then, we apply the trained models on other graphs directly without fine tuning.

As shown in Table 5, DeepGraphlet has a good transfer ability. For example, models trained on sampled graphs of Orkut perform well on Artist and LJ. They outperform the models trained on sampled graphs of Artist when tested on Artist, and achieve a quite similar performance of the models trained on sampled graphs of LJ and tested on LJ. The experimental results also provide insights for how to choose a suitable training dataset. The transfer performance mainly depends on graph scales and graph types of trained and tested graphs.

For graph scales, the models trained on larger graphs perform better on the tested graphs. For instance, the models trained on Orkut outperform the models trained on LJ when tested on Artist and Topcats. Orkut has a much larger edge scale than LJ. Thus, DeepGraphlet is more likely to capture more general graph structure information from sampled graphs of Orkut than LJ, which leads to better performance.

For graph types, the models transfer better on the graphs of the same type. In Table 5, Artist, LJ, and Orkut are social graphs, while Topcats is a web graph. When tested on Topcats, the models trained on sampled graphs of LJ and Orkut have worse performances than those trained on Topcats. As a web graph, the structures of Topcats are pretty different from that of social networks, which leads to such results. However, between the same type of graphs, DeepGraphlet transfers well. The models trained on sampled graphs of LJ and Orkut outperform the models trained on those of Artist when tested on Artist.

In summary, our DeepGraphlet has a great transfer ability, and we can pretrain DeepGraphlet on graphs that contain abundant graph structures (a fairly large graph scale) to obtain good results on unseen graphs.

Data. / Overall	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
DeepGraphlet-K	0.736±0.085	0.526±0.094	1.689±0.122	1.254±0.133	1.337±0.121	0.242±0.019	0.584±0.035	0.688±0.065
DeepGraphlet-R	0.514±0.105	0.278±0.015	0.563±0.010	0.673±0.035	0.492±0.027	0.098±0.002	0.361±0.029	0.357±0.009
DeepGraphlet-M	0.275±0.034	**0.145±0.008**	0.323±0.022	0.306±0.008	**0.352±0.010**	**0.059±0.001**	0.181±0.005	0.177±0.008
DeepGraphlet	**0.251±0.030**	0.151±0.011	**0.273±0.013**	**0.292±0.009**	0.386±0.015	0.061±0.002	**0.180±0.008**	**0.161±0.005**
Data. / 3 - LGF	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
DeepGraphlet-K	0.038±0.019	0.025±0.012	0.042±0.007	0.012±0.004	0.060±0.018	0.020±0.006	0.035±0.011	0.016±0.002
DeepGraphlet-R	0.018±0.009	0.005±0.000	0.013±0.001	0.008±0.002	0.007±0.001	0.008±0.000	0.009±0.001	0.003±0.000
DeepGraphlet-M	**0.009±0.007**	**0.003±0.000**	0.010±0.001	**0.003±0.000**	**0.005±0.001**	**0.004±0.000**	**0.002±0.000**	0.002±0.000
DeepGraphlet	0.010±0.006	0.004±0.001	**0.006±0.001**	0.004±0.001	**0.005±0.001**	**0.004±0.000**	**0.002±0.000**	**0.001±0.000**
Data. / 4 - LGF	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
DeepGraphlet-K	0.277±0.037	0.198±0.038	0.483±0.041	0.492±0.073	0.423±0.024	0.094±0.010	0.189±0.034	0.201±0.026
DeepGraphlet-R	0.215±0.080	0.134±0.022	0.161±0.004	0.241±0.006	0.140±0.006	0.033±0.000	0.102±0.009	0.115±0.011
DeepGraphlet-M	0.105±0.010	0.049±0.006	0.073±0.009	0.101±0.006	**0.086±0.003**	**0.017±0.001**	0.040±0.003	0.048±0.007
DeepGraphlet	**0.094±0.013**	**0.045±0.004**	**0.057±0.003**	**0.069±0.004**	0.091±0.005	0.018±0.001	**0.037±0.003**	**0.036±0.002**
Data. / 5 - LGF	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut
DeepGraphlet-K	0.421±0.061	0.303±0.080	1.164±0.114	0.751±0.081	0.854±0.124	0.128±0.010	0.360±0.013	0.471±0.046
DeepGraphlet-R	0.281±0.053	0.138±0.012	0.388±0.007	0.424±0.033	0.345±0.023	0.057±0.001	0.250±0.026	0.238±0.004
DeepGraphlet-M	0.161±0.022	**0.092±0.011**	0.240±0.017	**0.202±0.009**	**0.261±0.013**	**0.038±0.001**	**0.139±0.006**	0.127±0.006
DeepGraphlet	**0.147±0.016**	0.102±0.009	**0.210±0.011**	0.219±0.005	0.290±0.012	0.039±0.001	0.142±0.005	**0.123±0.003**

Table 3: Ablation study that compares the effectiveness of DeepGraphlet and its variants measured in KL divergence. The smaller value means better performance

Data / Alg.	Slashdot	Artist	Google	Topcats	BerkStan	Patents	LJ	Orkut	Friendster
EVOKE	4.5	121	285	7.2K	3.6K	1K	20.3K	139.7K	——
motivo	170.2	90.1	3.8K	26.1K	34.6K	6.3K	25.8K	42.6K	362.3K
k-tuple feature	1	1	12	11	9	52	64	82	1.8K
DeepGraphlet-M	1.8	1.4	14.2	56.6	11.8	69.8	90.9	199.2	10.8K
DeepGraphlet	0.9	0.6	7.0	28.3	5.8	41.3	53.7	104.5	4.3K

Table 4: Running time of models for computing 3-, 4- and 5-order LGF. Time is measured in the unit of second. The running time of other GNN baselines is similar to that of DeepGraphlet-M.

sumLGF / Train	Artist	Topcats	LJ	Orkut
Artist	0.151	0.576	0.237	0.453
Topcats	0.115	**0.292**	0.191	0.308
LJ	0.092	0.591	**0.181**	0.248
Orukt	**0.090**	0.585	0.189	**0.161**

Table 5: Results of training and validating DeepGraphlet on sampled small graphs of one graph and then testing the model's performance (KL divergence) on other graphs. The smaller value means better performance.

5.4 Results of Efficiency

In this section, we present the evaluation result of model efficiency (See Table 4). Our proposed framework's computation of LGF on new graphs comprises two parts: k-tuple feature extraction time and GNN model running time.

DeepGraphlet achieves a significant improvement of computational speed compared to EVOKE and motivo. For instance, on the Orkut (hundreds of million-scale graphs), DeepGraphlet achieves a 749x speedup compared to EVOKE. When it comes to Friendster (a billion-scale graph), EVOKE cannot handle it, while motivo spends 59 times running time than DeepGraphlet. This result of DeepGraphlet's running time is consistent with the analysis presented in Section 4.3. DeepGraphlet's time complexity is linearly related to a graph's scale.

The difference between DeepGraphlet and DeepGraphlet-M is the multitask mechanism. DeepGraphlet is a 3-layer GNN with one output layer after each GNN layer. DeepGraphlet-M uses three

GNN models: 1-, 2- and 3-layer GNN to compute 3-, 4- and 5-LGF respectively. Therefore, DeepGraphlet achieves a 2x speedup relative to DeepGraphlet-M in terms of running time, which demonstrates the effectiveness of the multi-task mechanism.

6 CONCLUSION

In this paper, we address the challenging LGF computing problem. We are the first to investigate the power of GNN for LGF computation on billion-scale graphs. We propose a novel model, DeepGraphlet. The k-tuple features enable GNNs to capture rich structural information efficiently. Furthermore, we theoretically prove its expressive power. DeepGraphlet captures the cross- and inner-order relationships among graphlets. At the same time, a multi-task mechanism enables the model to learn different-order graphlets simultaneously. Experiments show that DeepGraphlet achieves promising effectiveness improvements relative to baselines. Moreover, the model's time complexity is linearly related to the number of nodes and edges in a graph. DeepGraphlet achieves a 749x speedup when compared with exact graphlet computing algorithms on hundreds of millions-scale graphs. Furthermore, DeepGraphlet can handle billion-scale graphs, achieving 59x speedup than sampling baselines.

REFERENCES

[1] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. (2015), 1–10.
[2] Nesreen K Ahmed, Theodore L Willke, and Ryan A Rossi. 2016. Estimation of local subgraph counts. (2016), 586–595.
[3] Andrew Baas, Frances Hung, Hao Sha, Mohammad Al Hasan, and George Mohler. 2018. Predicting virality on networks using local graphlet frequency Distribution. (2018), 2475–2482.
[4] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. Simgnn: A neural network approach to fast graph similarity computation. (2019), 384–392.
[5] Yunsheng Bai, Hao Ding, Ken Gu, Yizhou Sun, and Wei Wang. 2020. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. (2020), 3219–3226.
[6] Suman K Bera and C Seshadhri. 2020. How to count triangles, without seeing the whole graph. (2020), 306–316.
[7] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. (2012), 91–100.
[8] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. 2020. Improving graph neural network expressivity via subgraph isomorphism counting. (2020).
[9] Marco Bressan, Stefano Leucci, and Alessandro Panconesi. 2019. Motivo: Fast motif counting via succinct color coding and adaptive sampling. *PVLDB* 12, 11 (2019), 1651–1663.
[10] Xiaowei Chen and John CS Lui. 2017. A unified framework to estimate global and local graphlet counts for streaming graphs. (2017), 131–138.
[11] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. 33 (2020), 10373–10385.
[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. (2016), 3844–3852.
[13] Changjun Fan, Li Zeng, Yuhui Ding, Muhao Chen, Yizhou Sun, and Zhong Liu. 2019. Learning to identify high betweenness centrality nodes from scratch: A novel graph neural network approach. (2019), 559–568.
[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. (2017), 1024–1034.
[15] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.
[16] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. (2016).
[17] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. 6, 1 (2009), 29–123.
[18] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. 33 (2020).
[19] Yongsub Lim and U Kang. 2015. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. (2015), 685–694.
[20] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. 2020. Neural subgraph isomorphism counting. (2020), 1959–1969.
[21] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. 2019. Fast approximations of betweenness centrality with graph neural networks. (2019), 2149–2152.
[22] Tijana Milenković, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. 2010. Optimal network alignment with graphlet degree vectors. 9 (2010), CIN–S4744.
[23] Tijana Milenković and Nataša Pržulj. 2008. Uncovering biological network function via graphlet degree signatures. 6 (2008), CIN–S680.
[24] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: Simple building blocks of complex networks. 298, 5594 (2002), 824–827.
[25] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. 33, 01 (2019), 4602–4609.
[26] Kirill Paramonov, Dmitry Shemetov, and James Sharpnack. 2019. Estimating graphlet statistics via lifting. (2019), 587–595.
[27] Noujan Pashanasangi and C Seshadhri. 2020. Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke. (2020), 447–455.
[28] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. (2017), 1431–1440.
[29] Jiaxuan You Rex Ying, Andrew Z. Wang and Jure Leskovec. 2020. Frequent subgraph mining by walking in order embedding space. (2020).
[30] Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. 2019. A survey on subgraph counting: Concepts, algorithms and applications to network motifs and graphlets. (2019).
[31] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. GEMSEC: Graph embedding with self clustering. (2019), 65–72.
[32] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. 2020. Random features strengthen graph neural networks. (2020).
[33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. 20, 1 (2008), 61–80.
[34] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. 11, 4 (2017), 1–50.
[35] Behrooz Tahmasebi and Stefanie Jegelka. 2020. Counting substructures with higher-order graph neural networks: possibility and impossibility results. (2020).
[36] Ata Turk and Duru Turkoglu. 2019. Revisiting wedge sampling for triangle counting. (2019), 1875–1885.
[37] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. 2011. The anatomy of the facebook social graph. (2011).
[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. (2017).
[39] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. 393, 6684 (1998), 440–442.
[40] Sebastian Wernicke. 2005. A faster algorithm for detecting network motifs. (2005), 165–177.
[41] Sebastian Wernicke and Florian Rasche. 2006. FANMOD: A tool for fast network motif detection. *Bioinformatics* 22, 9 (2006), 1152–1153.
[42] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? (2018).
[43] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. (2015), 1365–1374.
[44] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. 42, 1 (2015), 181–213.
[45] Yang Yang, Yuhong Xu, Chunping Wang, Yizhou Sun, Fei Wu, Yueting Zhuang, and Ming Gu. 2019. Understanding default behavior in online lending. (2019), 2043–2052.
[46] Hao Yin, Austin R Benson, and Jure Leskovec. 2018. Higher-order clustering in networks. 97, 5 (2018), 052306.
[47] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. (2017), 555–564.

A PROOF OF THEOREMS

Before addressing the detailed proof of the theorems, we outline lemmas and the 1-WL in Algorithm 1 as preliminaries. Here, $\{\{...\}\}$ denotes a multiset, while the function $HASH$ is injective.

Definition A.1 (Corresponding nodes pair). Graph isomorphism problem is determining whether there is a node mapping between two graphs that the two graphs are the same after the mapping. If G and G' are isomorphic, $v \in G$ and $v' \in G'$, v are mapped to v' in the mapping that make the two graphs isomorphic, then we call (v, v') corresponding nodes pair.

The following two lemmas are for the proof of Theorems 4.1 and Theorem 4.2.

Lemma A.2. *Graphs G and G' are isomorphic, and in all corresponding nodes pairs, the features of the two nodes are the same. GNNs can output the same representation for G and G', and determine that the two graphs are isomorphic.*

Proof. Due to G and G' being isomorphic, 1-WL will output the same representation for the two graphs. GNNs' computation is the same as 1-WL, except that GNNs use neural networks in the aggregate and update functions and 1-WL uses the injective hash function. Xu [42] proves that the computation process of GNNs with appropriate aggregate and update functions can be injective, then GNNs can also output the same representation for G and G'. Therefore, GNNs identify the two graphs are isomorphic. □

Lemma A.3. *Given a pair of graphs $G = (V, E, X)$ and $G' = (V', E', X')$, and $\{\{x_v^0 | v \in V\}\} \neq \{\{x_{v'}'^0 | v' \in V'\}\}$, then GNNs can output different representations for G and G', and distinguish the two graphs.*

Proof. As $\{\{x_v^{l-1} | v \in V\}\} \neq \{\{x_{v'}'^{l-1} | v' \in V'\}\}$ and all the computation of GNNs with appropriate aggregate and update functions can be injective, then the final representation of two graphs are different no matter what the graph structure is. Thus, GNNs can distinguish these two graphs. □

Algorithm 1 1-WL

Input: A pair of graphs $G = (V, E, X)$ and $G' = (V', E', X')$
Output: Whether two graphs G and G' are isomorphic

$c_v^0 = HASH(X_v)(\forall v \in V)$
$c_{v'}'^0 = HASH(X_{v'}')(\forall v' \in V')$
$l = 0$;
while not converging:
    l += 1
    if $\{\{c_v^{l-1} | v \in V\}\} \neq \{\{c_{v'}'^{l-1} | v' \in V'\}\}$
        return 'not isomorphic'
    $c_v^l = HASH(c_v^{l-1}, \{\{c_u^{l-1} | u \in N(v)\}\})(\forall v \in V)$
    $c_{v'}'^l = HASH(c_{v'}'^{l-1}, \{\{c_{u'}'^{l-1} | u' \in N(v')\}\})(\forall v' \in V')$
return 'isomorphic';

A.1 Proof of Theorem 4.1

Proof. Suppose we have two graphs $G = (V, E, X)$ and $G' = (V', E', X')$. K and K' are the extracted k-tuple features of the two graphs' nodes. $F = X||K$ and $F' = X'||K'$ represent the node feature initialization of GNNs with k-tuple features, where $||$ denotes concatenate.

Case 1. G and G' are isomorphic, and 1-WL always regards their graph structures as the same. For the k-tuple feature, given that the structure of the two graphs is the same, the isomorphic types of the sampled tuples of the two graphs follow the same distribution. According to the Law of Large Numbers, when the sample times approaches infinity, the k-tuple features will converge to the same. $(X = X' \text{ and } K = K') \Rightarrow F = F'$, by applying Lemma A.2, GNN can identify that G and G' are isomorphic. Therefore, GNNs with k-tuple features have the same expressive power as 1-WL in this case from a probabilistic perspective.

Case 2. G and G' are not isomorphic, and 1-WL regards their graph structures differently. It means that in the 1-WL computation, at least in one iteration, there is a corresponding nodes pair (v, v') that the two nodes' representations are different. Because the computation of GNNs can be injective, with X (1-WL's initial feature) as node feature initialization, GNNs can output different representations for the two nodes in the corresponding nodes pair (v, v'). Then if we feed GNNs with $X||K$, the representations of the two nodes (v, v') are also different due to the injective property of GNN. Thus, GNNs can also determine that these two graphs are different, and in this case, the two algorithms have equal ability.

Case 3. G and G' are not isomorphic and 1-WL regards their graph structure as same. 1-WL cannot distinguish specific graph structures even though they are different [11]. However, GNNs with k-tuple features can distinguish more pairs of graphs than 1-WL, as their awareness of specific high-order structures, such as Figure 3 and Figure 5. According to Lemma A.3, GNN can identify these graphs are non-isomorphic. Thus, GNNs with k-tuple features is a more powerful approach than 1-WL in this case.

The above three cases cover all possible scenarios. In summary, GNNs with k-tuple features, can exceed the expressive power of 1-WL in the graph isomorphism problem. □

A.2 Proof of Theorem 4.2

Proof. We prove that (k-1)-WL algorithm is not stronger than GNNs with k-tuple features by showing that there are graphs that (k-1)-WL cannot distinguish while GNNs with k-tuple features can. As shown in Figure 5(a), 1-WL and 2-WL cannot distinguish these two graphs. However, GNNs with the k-tuple feature are aware of the triangles' existence. And in Figure 5(b), [8] says that 3-WL cannot distinguish the two graphs. As 4-clique exist in one graph but not in another, the k-tuple features of the two graphs are different.

$$\{\{k_v^{l-1} | v \in V\}\} \neq \{\{k_{v'}'^{l-1} | k' \in V'\}\} \Rightarrow F \neq F'$$

According to Lemma A.3, GNNs with k-tuple features can distinguish these two graphs. In conclusion, (k-1)-WL algorithm is not stronger than GNNs with k-tuple features in graph isomorphism problem where $3 \leq k \leq 4$. □

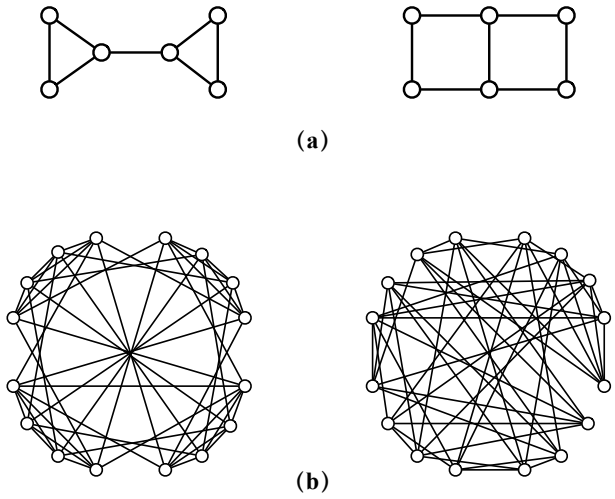

Figure 5: Example of 1-, 2- and 3-WL fails.

B K-TUPLE FEATURE PSEUDO CODE

The pseudo-code for k-tuple feature generation is presented in Algorithm 2.

Algorithm 2 k-tuple feature extraction.

Input: Graph $G = (V, E)$, the sample number of k-tuples for each node, n; the order of sampled k-tuple, k; the list of all non-isomorphism k-tuple graph sets, S

Output: k-tuple feature K with shape($|V|$, $|S|$);

Construct variables for Alias sampling methods for random drawing a node's neighbor in O(1);
Fill K shape($|V|$, $|S|$) with zeros;
for v in V:
    for i in range(n):
        $T = [v]$
        for j in range($k - 1$):
            randomly sample a node u in T with probability proportional to their degree;
            randomly sample a neighbor $n(u)$ of node u with probability proportional to neighbors' degree;
            add the neighbor $n(u)$ to T;
        Determine the induced subgraph of T is isomorphic to which k-tuple in S, and add the corresponding value in k_v by $1/n$
return K;

C IMPLEMENTATION DETAILS

All experiments are conducted on a 56-core CPU with 384GB memory and NVIDIA RTX 2080 GPUs. Since EVOKE only utilizes CPUs, neural models are trained with GPUs and tested on CPUs to facilitate fair comparison. Each experiment is repeated five times.

We add dropout with a drop probability 0.5 and a batch normalization after each layer for neural models. All embedding dimensions and hidden embedding dimensions are set to 128. The Adam optimizer with a learning rate of 0.001 is adopted to train the models. The layer numbers of GNNs with k-tuple features are set to 3, while the others are set to 5. For models without multitask mechanism (DeepGraphlet-M), if the number of GNN layers is set to L, we train three different GNN models of $L-2$, $L-1$ and L layers to compute 3-, 4- and 5-order LGF respectively. In all experiments, for k-tuple features, we sample 3-, 4- and 5-tuple 100 times for every node.