Graph Generative Model for Benchmarking Graph Neural Networks

Minji Yoon [1] **Yue Wu** [1] **John Palowitch** [2] **Bryan Perozzi** [2] **Russ Salakhutdinov** [1]

Abstract

As the field of Graph Neural Networks (GNN) continues to grow, it experiences a corresponding increase in the need for large, real-world datasets to train and test new GNN models on challenging, realistic problems. Unfortunately, such graph datasets are often generated from online, highly privacy-restricted ecosystems, which makes research and development on these datasets hard, if not impossible. This greatly reduces the amount of benchmark graphs available to researchers, causing the field to rely only on a handful of publicly-available datasets. To address this problem, we introduce a novel graph generative model, Computation Graph Transformer (CGT) that learns and reproduces the distribution of real-world graphs in a privacy-controlled way. More specifically, CGT (1) generates effective benchmark graphs on which GNNs show similar task performance as on the source graphs, (2) scales to process large-scale graphs, (3) incorporates off-the-shelf privacy modules to guarantee end-user privacy of the generated graph. Extensive experiments across a vast body of graph generative models show that only our model can successfully generate privacy-controlled, synthetic substitutes of large-scale real-world graphs that can be effectively used to benchmark GNN models.

1. Introduction

Graph Neural Networks (GNNs) (Kipf & Welling, 2016a; Chami et al., 2022) are machine learning models that learn the dependences in graphs via message passing between nodes. Various GNN models have been widely applied on a variety of industrial domains such as misinformation detection (Benamira et al., 2019), financial fraud detection (Wang et al., 2019), traffic prediction (Zhao et al., 2019), and social recommendation (Ying et al., 2018). However, datasets from these industrial tasks are overwhelmingly proprietary and privacy-restricted and thus almost always unavailable for researchers to study or evaluate new GNN architectures. This state-of-affairs means that in many cases, GNN models cannot be trained or evaluated on graphs that are appropriate for the actual tasks that they need to execute.

In this paper, we propose a novel graph generation problem to overcome the limited access to real-world graph datasets. Given a graph, our goal is to generate synthetic graphs that follow its distribution in terms of graph structure, node attributes, and labels, making them usable as substitutes for the original graph for GNN research. Any observations or results from experiments on the original graph should be near-reproduced on the synthetic graphs. Additionally, the graph generation process should be scalable and privacy-controlled to consume large-scale and privacy-restricted real-world graphs. Formally, our new graph generation problem is stated as follow:

Problem Definition 1. *Let $\mathcal{A}$, $\mathcal{X}$, and $\mathcal{Y}$ denote adjacency, node attribute, and node label matrices; given an original graph $\mathcal{G} = (\mathcal{A}, \mathcal{X}, \mathcal{Y})$, generate a synthetic graph dataset $\mathcal{G}'$ satisfying:*

- ***Benchmark effectiveness:*** *performance rankings among m GNN models on $\mathcal{G}'$ should be similar to the rankings among the same m GNN models on $\mathcal{G}$.*
- ***Scalability:*** *computation complexity of graph generation should be linearly proportional to the size of the original graph $O(|\mathcal{G}|)$ (i.e., number of nodes or edges).*
- ***Privacy guarantee:*** *any syntactic privacy notions are given to end users (e.g., k-anonymity).*

While there is already a vast body of work on graph generation, we found that no study has fully addressed the problem setting above. (Leskovec et al., 2010; Palowitch et al., 2022) generate random graphs using a few known graph patterns, while (You et al., 2018; Liao et al., 2019) learn only graph structures without considering node attribute/label information. Recent graph generative models (Shi et al., 2020; Luo et al., 2021) are mostly specialized to small-scale molecule graph generation.

In this work, we introduce a novel graph generative model, Computation Graph Transformer (CGT) that addresses the three requirements above for the benchmark graph gen-

[1]Carnegie Mellon University [2]Google Research. Correspondence to: Minji Yoon <minjiy@cs.cmu.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

eration problem. First, we reframe the graph generation problem into a discrete-value sequence generation problem. Motivated by GNN models that avoid scalability issues by operating on egonets sampled around each node, called *computation graphs* (Hamilton et al., 2017), we learn the distribution of *computation graphs* rather than the whole graph. In other words, our generated graph dataset $\mathcal{G}'$ will have a form of *a set of computation graphs* where GNN models can run immediately without preceded egonet sampling process. In addition to the scalability benefit, learning distributions of computation graphs which are the direct input to GNN models may also help to get better benchmark effectiveness. Then, instead of learning the joint distribution of graph structures and node attributes, we devise a novel *duplicate encoding* scheme for computation graphs that transforms an adjacency and feature matrix pair into a single, dense feature matrix that is isomorphic to the original pair. Finally, we quantize the feature matrix into a discrete value sequence that will be consumed by a Transformer architecture (Vaswani et al., 2017) adapted to our graph generation setting. After the quantization, our model can be easily extended to provide k-anonymity or differential privacy guarantees on node attributes and edge distributions by incorporating off-the-shelf privacy modules.

Extensive experiments on real-world graphs with a diverse set of GNN models demonstrate CGT provides significant improvement over existing generative models in terms of benchmark effectiveness (up to 1.03 higher Spearman correlations, up to 33% lower MSE between original and reproduced GNN accuracies), scalability (up to 35k nodes and 8k node attributes), and privacy guarantees (k-anonymity and differential privacy for node attributes). CGT also preserves graph statistics on computation graphs by up to 11.01 smaller Wasserstein distance than previous approaches.

In sum, our contributions are: 1) a novel graph generation problem featuring three requirements of modern graph learning; 2) reframing of the graph generation problem into a discrete-valued sequence generation problem; 3) a novel Transformer architecture able to encode the original computation graph structure in sequence learning; and finally 4) comprehensive experiments that evaluate the effectiveness of graph generative models to benchmark GNN models.

2. Related Work

Traditional graph generative models extract common patterns among real-world graphs (e.g. nodes/edge/triangle counts, degree distribution, graph diameter, clustering coefficient) (Chakrabarti & Faloutsos, 2006) and generate synthetic graphs following a few heuristic rules (Erdős et al., 1960; Leskovec et al., 2010; Leskovec & Faloutsos, 2007; Albert & Barabási, 2002). However, they cannot generate unseen patterns on synthetic graphs (You et al., 2018). More importantly, most of them generate only graph structures, sometimes with low-dimensional boolean node attributes (Eswaran et al., 2018). **General-purpose deep graph generative models** exploit GAN (Goodfellow et al., 2014), VAE (Kingma & Welling, 2013), and RNN (Zaremba et al., 2014) to learn graph distributions (Guo & Zhao, 2020). Most of them focus on learning graph structures (You et al., 2018; Liao et al., 2019; Simonovsky & Komodakis, 2018; Grover et al., 2019), thus their evaluation metrics are graph statistics such as orbit counts, degree coefficients, and clustering coefficients which do not consider quality of generated node attributes and labels. **Molecule graph generative models** are actively studied for generating promising candidate molecules using VAE (Jin et al., 2018), GAN (De Cao & Kipf, 2018), RNN (Popova et al., 2019), and recently invertible flow models (Shi et al., 2020; Luo et al., 2021). However, most of their architectures are specialized to small-scaled molecule graphs (e.g., 38 nodes per graph in the ZINC datasets) with low-dimensional attribute space (e.g., 9 node attributes indicating atom types) and distinct molecule-related information (e.g., SMILES representation or chemical structures such as bonds and rings) (Suhail et al., 2021).

3. From Graph Generation to Sequence Generation

In this section, we illustrate how to convert the whole-graph generation problem into a discrete-valued sequence generation problem. An input graph $\mathcal{G}$ is given as a triad of adjacency matrix $\mathcal{A} \in \mathbb{R}^{n \times n}$, node attribute matrix $\mathcal{X} \in \mathbb{R}^{n \times d}$, and node label matrix $\mathcal{Y} \in \mathbb{R}^{n}$ with n nodes and d-dimensional node attribute vectors.

3.1. Computation graph sampling in GNN training

Given large-scale real-world graphs, instead of operating on the whole graph, GNNs extract each node v's egonet $\mathcal{G}_v$, namely a *computation graph*, then compute embeddings of node v on $\mathcal{G}_v$. This means that in order to benchmark GNN models, we are not necessarily required to learn the distribution of the whole graph; instead, we can learn the distribution of computation graphs which are the direct input to GNN models. As with the global graph, a computation graph $\mathcal{G}_v$ is composed of a sub-adjacency matrix $\mathcal{A}_v \in \mathbb{R}^{n_v \times n_v}$, a sub-feature matrix $\mathcal{X}_v \in \mathbb{R}^{n_v \times d}$, and node v's label $\mathcal{Y}_v \in \mathbb{R}$, where each of n_v rows correspond to nodes sampled into the computation graph. Our problem then reduces to: *given a set of computation graphs* $\{\mathcal{G}_v = (\mathcal{A}_v, \mathcal{X}_v, \mathcal{Y}_v) : v \in \mathcal{G}\}$ *sampled from an original graph, we generate a set of computation graphs* $\{\mathcal{G}'_v = (\mathcal{A}'_v, \mathcal{X}'_v, \mathcal{Y}'_v)\}$. This reframing allows the graph generation process to scale to large-scale graphs.

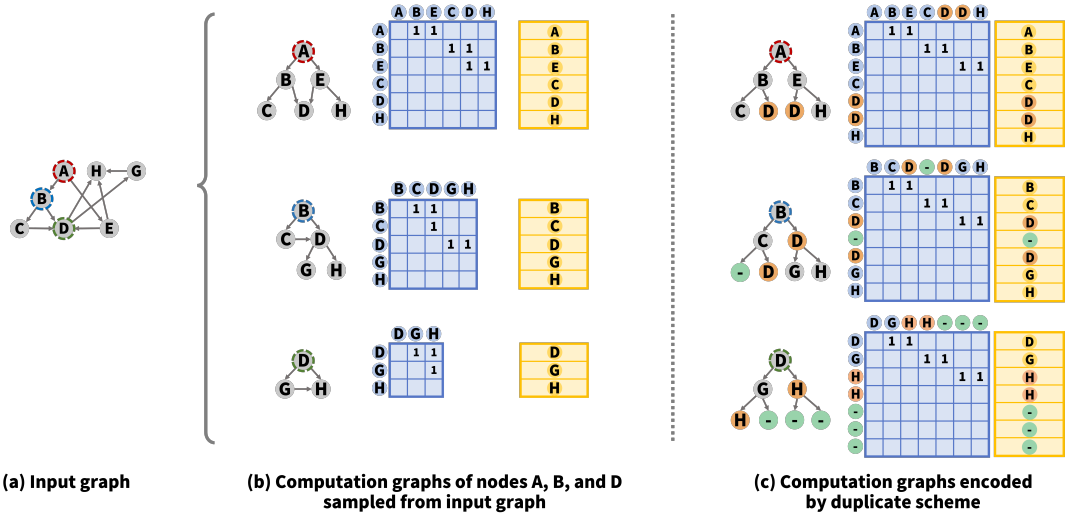


Figure 1: **Computation graphs with $s = 2$ neighbor samples and $L = 2$ depth:** (a) input graph; (b) original computation graphs have differently-shaped adjacency (blue) and attribute (yellow) matrices; (c) duplicate encoding scheme outputs the *same adjacency matrix* and *identically-shaped* attribute matrices.

3.2. Duplicate encoding scheme for computation graphs

Various sampling methods have been proposed to decide which neighboring nodes to add to a computation graph $\mathcal{G}_v$ given a target node v (Hamilton et al., 2017; Chen et al., 2018; Huang et al., 2018; Yoon et al., 2021). Two common rules across these sampling methods are 1) the number of neighbors sampled for each node is limited to keep computation graphs small and 2) the maximum distance (i.e., maximum number of hops) from the target node v to sampled nodes is decided by the depth of GNN models. Details on how to sample computation graphs can be found in Appendix A.3. This maximum number of neighbors is called the neighbor sampling number s and the maximum number of hops from the target node is called the depth of computation graphs L. Figure 1(b) shows computation graphs of nodes A, B, and D sampled with sampling number $s = 2$ and depth $L = 2$. Note that the shapes of computation graphs are variable.

Here we introduce a *duplicate encoding* scheme for computation graphs that is conceptually simple but brings a significant consequence: it *fixes the structure of all computation graphs* to the L-layered s-nary tree structure, allowing us to model all adjacency matrices as a constant. Starting from the target node v as a root node, we sample s neighbors iteratively L times from the computation graph. When a node has fewer neighbors than s, the duplicate encoding scheme defines a null node with zero attribute vector (node '$-$' in node B and D's computation graphs in Figure 1(c)) and samples it as a padding neighbor. When a node has a neighbor also sampled by another node, the duplicate encoding scheme copies the shared neighbor and provides each copy to parent nodes (node D in node A's computation graph is copied in Figure 1(c)). Each node attribute vector is also copied and added to the feature matrix. As shown in Figure 1(c), the duplicate encoding scheme ensures that all computation graphs have an identical adjacency matrix (presenting a balanced s-nary tree) and an identical shape of feature matrices. Under the duplicate encoding scheme, the graph structure information is fully encoded into feature matrices, which we will explain in details in Section 5.3. Note that in order to fix the adjacency matrix, we need to fix the order of nodes in adjacency and attribute matrices (e.g., breadth-first ordering in Figure 1(c)).

Now our problem reduces to learning the distribution of (duplicate-encoded) feature matrices of computation graphs: *given a set of feature matrix-label pairs* $\{(\tilde{\mathcal{X}}_v, \mathcal{Y}_v) : v \in \mathcal{G}\}$ *of duplicate-encoded computation graphs, we generate a set of feature matrix-label pairs* $\{(\tilde{\mathcal{X}}'_v, \mathcal{Y}'_v)\}$.

3.3. Quantization

To learn the distribution of feature matrices of computation graphs, we quantize feature vectors into discrete bins; specifically, we cluster feature vectors in the original graph using k-means and map each feature vector to its cluster id. Quantization is motivated by 1) privacy benefits and 2) ease of modeling. By mapping different feature vectors (which are clustered together) into the same cluster id, we can guarantee k-anonymity among them (more details in Section 4.2). Ultimately, quantization further reduces our problem to *learning the distribution of sequences of discrete values*, namely the sequences of cluster ids of feature vectors in each computation graph. Such a problem is naturally addressed by Transformers, state-of-the-art sequence generative models (Vaswani et al., 2017). In Section 4, we introduce the Computational Graph Transformer (CGT), a novel architecture which learns the distribution of computa-

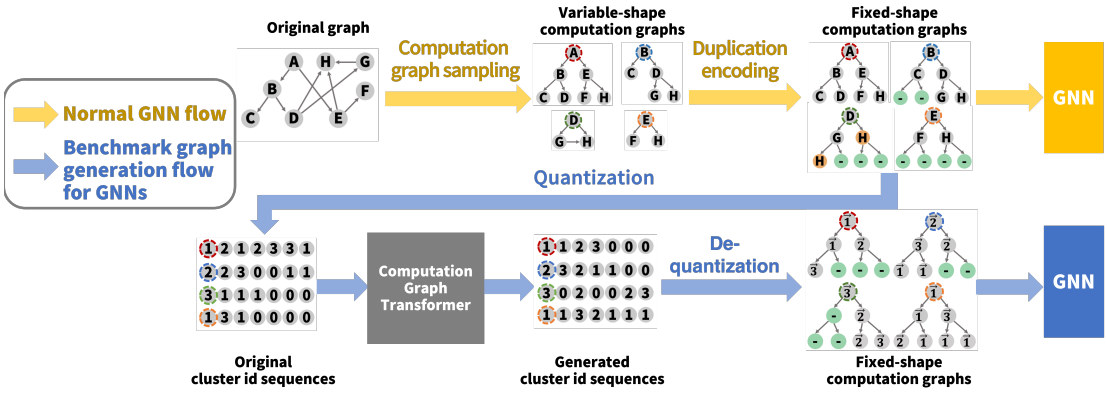


Figure 2: **Overview of our benchmark graph generation framework:** (1) We sample a set of computation graphs of variable shapes from the original graph, then (2) duplicate-encode them to fix adjacency matrices to a constant. (3) Duplicate-encoded feature matrices are quantized into cluster id sequences and fed into our Computation Graph Transformer. (4) Generated cluster id sequences are de-quantized back into duplicate-encoded feature matrices and fed into GNN models with the constant adjacency matrix.

tion graph structures encoded in the sequences effectively.

3.4. End-to-end framework for a benchmark graph generation problem

Figure 2 summarizes the entire process of mapping a graph generation problem into a discrete sequence generation problem. In the training phase, we 1) sample a set of computation graphs from the input graph, 2) encode each computation graph using the duplicate encoding scheme to fix adjacency matrices, 3) quantize feature vectors to cluster ids they belong to, and finally 4) hand over a set of *(sequence of cluster ids, node label)* pairs to our new Transformer architecture to learn their distribution. In the generation phase, we follow the same process in the opposite direction: 1) the trained Transformer outputs a set of *(sequence of cluster ids, node label)* pairs, 2) we de-quantize cluster ids back into the feature vector space by replacing them with the mean feature vector of the cluster, 3) we regenerate a computation graph from each sequence of feature vectors with the adjacency matrix fixed by the duplicate encoding scheme, and finally 4) we feed the set of generated computation graphs into the GNN model we want to train or evaluate.

4. Model

We present the Computation Graph Transformer that encodes the computation graph structure into sequence generation process with minimal modification to the Transformer architecture. Then we check our model satisfies the privacy and scalability requirements from Problem Definition 1.

4.1. Computation Graph Transformer (CGT)

In this work, we extend a two-stream self-attention mechanism, XLNet (Yang et al., 2019), which modifies the Transformer architecture (Vaswani et al., 2017) with a causal self-attention mask to enable auto-regressive generation. Given a sequence $\mathbf{s} = [s_1, \cdots, s_T]$, the M-layered Transformer maximizes the likelihood under the forward auto-regressive factorization as follows:

$$\max_{\theta} \log p_\theta(\mathbf{s}) = \sum_{t=1}^{T} \log p_\theta(s_t|\mathbf{s}_{<t}) = \sum_{t=1}^{T} \log \frac{exp(q_\theta^{(L)}(\mathbf{s}_{1:t-1})^\top e(s_t))}{\sum_{s' \neq s_t} exp(q_\theta^{(L)}(\mathbf{s}_{1:t-1})^\top e(s'))}$$

where token embedding $e(s_t)$ maps discrete input id s_t to a randomly initialized trainable vector, and query embedding $q_\theta^{(L)}(\mathbf{s}_{1:t-1})$ encodes information until $(t-1)$-th token in the sequence. More details on the XLNet architecture can be found in the Appendix A.12. Here we describe how we modify XLNet to encode computation graphs effectively.

Position embeddings: In the original Transformer architecture, each token receives a position embedding encoding its position in the sequence. In our model, sequences are flattened computation graphs (the input computation graph in Figure 3(a) is flattened into input sequence in Figure 3(b)). To encode the original computation graph structure, we provide different position embeddings to different layers in the computation graph, while nodes at the same layer share the same position embedding. When $l(t)$ denotes the layer number where t-th node is located at the original computation graph, position embedding $p_{l(t)}$ indexed by the layer number is assigned to t-th node. In Figure 3(b), node C, D, F and H located at the 1-st layer in the computation graph have the same position embedding p_1.

Attention masks: In the original architecture, query and context embeddings, $q_t^{(l)}$ and $h_t^{(l)}$, attend to all context embeddings $\mathbf{h}_{1:t-1}^{(l-1)}$ before t. In the computation graph, each node is sampled based on its parent node (which is sampled

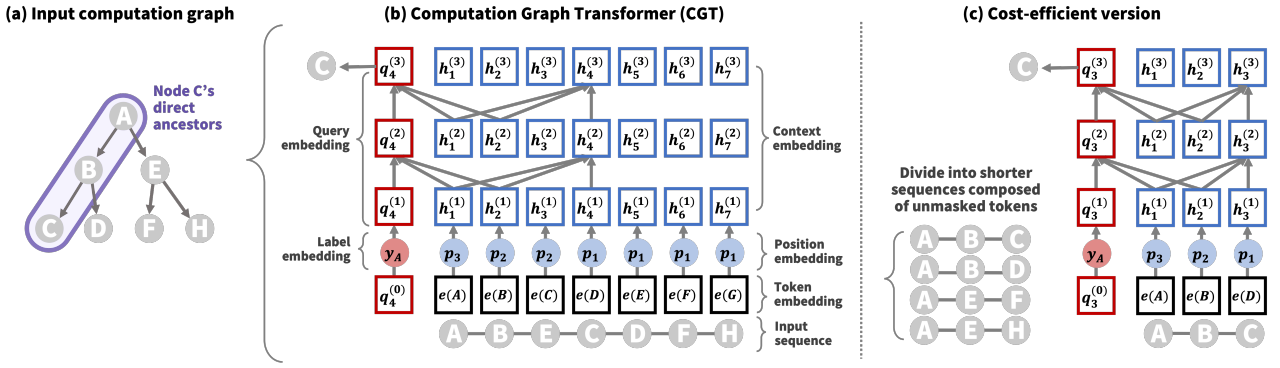


Figure 3: **Computation Graph Transformer (CGT):** (a,b) Given a sequence flattened from the input computation graph, CGT generates context in the forward direction. $e(s_t)$, $q_t^{(l)}$, and $h_t^{(l)}$ denote the token, query, and context embedding of t-th token at the l-th layer; $p_{l(t)}$ and y_{s_1} denote the position embeddings of t-th token and label embedding of the whole sequence, respectively. (c) The cost-efficient version of CGT divides the input sequence into shorter ones composed only of direct ancestor nodes.

based on its own parent nodes) and is not directly affected by its sibling nodes. To encode this relationship more effectively, we mask all nodes except direct ancestor nodes in the computation graph, i.e., the root node and any nodes between the root node and the leaf node. In Figure 3(b), node C's context/query embeddings attend only to direct ancestors, nodes A and B. Note that the number of unmasked tokens are fixed to L in our architecture because there are always $L-1$ direct ancestors in L-layered computation graphs. Based on this observation, we design a cost-efficient version of CGT that has shorter sequence length and preserves XLNet's auto-regressive masking as shown in Figure 3(c).

Label conditioning: Distributions of neighboring nodes are not only affected by each node's feature information but also by its label. It is well-known that GNNs improve over MLP performance by adding convolution operations that augment each node's features with neighboring node features. This improvement is commonly attributed to nodes whose feature vectors are noisy (outliers among nodes with the same label) but that are connected with "good" neighbors (whose features are well-aligned with the label). In this case, without label information, we cannot learn whether a node has feature-wise homogeneous neighbors or feature-wise heterogeneous neighbors but with the same label. In our model, query embeddings $q_t^{(0)}$ are initialized with label embeddings y_{s_1} that encode the label of the root node s_1.

4.2. Theoretical analysis

Our framework provides k-anonymity for node attributes and edge distributions by using k-means clustering with the minimum cluster size k (Bradley et al., 2000) during the quantization phase. Note that we define edge distributions as neighboring node distributions of each node. The full proofs for the following claims can be found in Appendix A.4.

Claim 1 (k-anonymity for node attributes and edge distributions)**.** *In the generated computation graphs, each node's attributes and edge distribution appear at least k times.*

We can also provide differential privacy (DP) for node attributes and edge distributions by exploiting DP k-means clustering (Chang et al., 2021) during the quantization phase and DP stochastic gradient descent (DP-SGD) (Song et al., 2013) to train the Transformer. Unfortunately, however, DP-SGD for Transformer networks doesn't yet work reliably in practice. Thus we cannot guarantee *strict* DP for edge distributions in practice (experimental results in Section 5.2.3 and more analysis in Appendix A.4). Thus, here, we claim DP only for node attributes.

Claim 2 ((ϵ, δ)-Differential Privacy for node attributes)**.** *With probability at least $1-\delta$, our generative model A gives ϵ-differential privacy for any graph $\mathcal{G}$, any neighboring graph $\mathcal{G}_{-v}$ without any node $v \in \mathcal{G}$, and any new computation graph $\mathcal{G}_{cg}$ generated from our model as follows:*

$$e^{-\epsilon} \leq \frac{Pr[A(\mathcal{G}) = \mathcal{G}_{cg}]}{Pr[A(\mathcal{G}_{-v}) = \mathcal{G}_{cg}]} \leq e^{\epsilon}$$

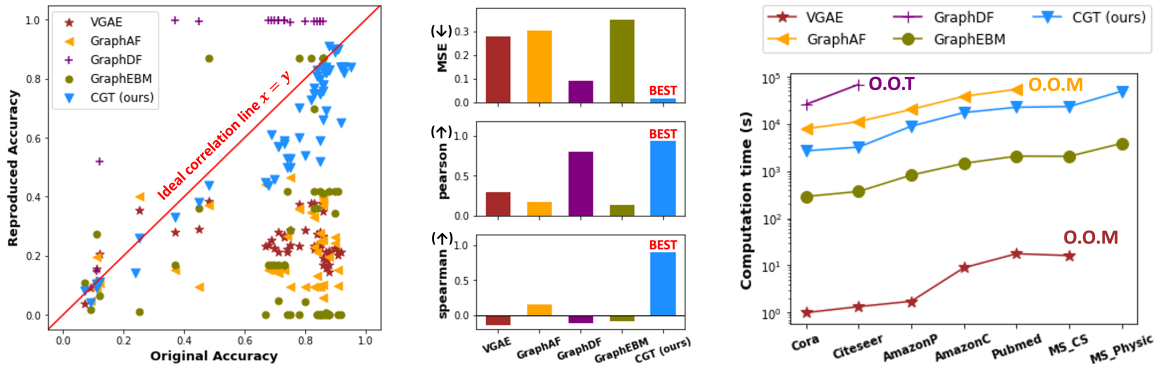
Finally, we show that CGT satisfies the scalability requirement in Problem Definition 1:

Claim 3 (Scalability)**.** *To generate L-layered computation graphs with neighbor sampling number s on a graph with n nodes, computational complexity of* CGT *training is $O(s^{2L}n)$, and the cost-efficient version is $O(L^2 s^L n)$.*

5. Experiments

5.1. Experimental setting

Baselines: We choose 5 state-of-the-art graph generative models that learn graph structures with node attribute information: two VAE-based general graph generative models, VGAE (Kipf & Welling, 2016b) and GraphVAE (Simonovsky & Komodakis, 2018) and three molecule



(a) Reproduced GNN accuracy (b) Benchmark effectiveness (c) Scalability

Figure 4: **Benchmark effectiveness and scalability in graph generation.** (a) We evaluate graph generative models by how well they reproduce GNN performance from the original graph (X-axis: original accuracy) on synthetic graphs (Y-axis: reproduced accuracy). Our method is closest to $x = y$, which is ideal. (b) We measure Mean Square Error (MSE) and Pearson/Spearman correlations from results in (a). Our method shows the lowest MSE and highest correlations. (c) We measure the computation time (training + evaluation) of each graph generative model. Only our method is scalable across all datasets while showing the best performance. O.O.T denotes out-of-time (> 20 hrs) and O.O.M denotes out-of-memory errors.

Table 1: **Privacy-Performance trade-off in graph generation**

	Original	**No privacy**	**K-anonymity**			**DP kmean** ($\delta = 0.01$)			**DP SGD** ($\delta = 0.1$)	
			$k = 100$	$k = 500$	$k = 1000$	$\epsilon = 1$	$\epsilon = 10$	$\epsilon = 25$	$\epsilon = 10^6$	$\epsilon = 10^9$
Pearson (↑)	1.000	0.934	0.916	0.862	0.030	0.874	0.844	0.804	0.112	0.890
Spearman (↑)	1.000	0.935	0.947	0.812	0.018	0.869	0.805	0.807	0.116	0.959

graph generative models, GraphAF (Shi et al., 2020), GraphDF (Luo et al., 2021), and GraphEBM (Suhail et al., 2021). While VGAE encodes the large-scale whole graph at once, the other 4 graph generative models are designed to process a set of small-sized graphs. Thus we provide the original whole graph to GVAE and a set of sampled computation graphs to the other baselines, respectively.

Datasets: We evaluate on 7 public datasets — 3 citation networks (Cora, Citeseer, and Pubmed) (Sen et al., 2008), 2 co-purchase graphs (AmazonC and AmazonP) (Shchur et al., 2018), and 2 co-authorship graph (MS CS and MS Physic) (Shchur et al., 2018). Note that these datasets are the largest ones the baselines have been applied on. Data statistics can be found in Appendix A.15.

GNN models: We choose 9 of the most popular GNN models for benchmarking: 4 GNN models with different aggregators, GCN (Kipf & Welling, 2016a), GIN (Xu et al., 2018), SGC (Wu et al., 2019), and GAT (Veličković et al., 2017), 4 GNN models with different sampling strategies, GraphSage (Hamilton et al., 2017), FastGCN (Chen et al., 2018), AS-GCN (Huang et al., 2018), and PASS (Yoon et al., 2021), and one GNN model with PageRank operations, PPNP (Klicpera et al., 2018). Descriptions of each GNN model can be found in the Appendix A.11.1.

5.2. Main results

In this experiment, each graph generative model learns the distributions of 7 graph datasets and generates synthetic graphs. Then we train and evaluate 9 GNN models on each pair of original and synthetic graphs, and measure Mean Square Error (MSE) and Pearson/Spearman correlations (Myers et al., 2013) between the GNN performance on each pair of graphs. As shown in Figure 4(a), each graph generative model compares up to 63 pairs of original and reproduced GNN performances. Unless additionally specified, K-anonymity is set to $K = 30$ across all experiments.

5.2.1. Benchmark effectiveness.

In Figure 4(b), our proposed CGT shows up to 33% lower MSE, 0.80 higher Pearson and 1.03 higher Spearman correlations than all baselines. GraphVAE fails to converge, thus omitted in Figure 4. This results clearly show the graph generative models specialized to molecules cannot be generalized to the large-scale graphs with a high-dimensional feature space. The predicted distributions by baselines sometimes collapse to generating the the same node feature/labels across all nodes (e.g., 0% or 100% accuracy for all GNN models in Figure 4(a)), which is obviously not the most effective benchmark.

Table 2: **Comparison with simple privacy baselines that add noisy nodes and edges to the original graph.** *Node/Edge re-ident. prob.* columns show node/edge re-identification probabilities of each privacy method. **-** denotes no privacy trick has applied.

Node attributes	Edge distribution	Node re-ident. prob. (↓)	Edge re-ident. prob. (↓)	GCN	SGC	GIN	GAT	MSE (↓)
-	Edge addition (×2)	100%	50%	0.82	0.82	0.80	0.55	0.021
	Edge addition (×10)	100%	10%	0.39	0.40	0.37	0.70	0.168
	Edge deletion (50%)	100%	50%	0.83	0.83	0.82	0.84	0.001
	Edge deletion (100%)	100%	0%	0.73	0.73	0.73	0.72	0.014
Noise addition (×5)	-	20%	100%	0.82	0.82	0.82	0.18	0.106
	Edge addition (×2)	20%	50%	0.67	0.67	0.68	0.07	0.169
	Edge addition (×10)	20%	10%	0.07	0.30	0.31	0.07	0.449
	Edge deletion (50%)	20%	50%	0.78	0.77	0.77	0.15	0.120
	Edge deletion (100%)	20%	0%	0.39	0.40	0.38	0.11	0.291
K-anonymity (5)	K-anonymity (5)	20%	20%	0.83	0.82	0.83	0.83	0.001
K-anonymity (100)	K-anonymity (100)	1%	1%	0.75	0.74	0.76	0.74	0.010
K-anonymity (500)	K-anonymity (500)	0.2%	0.2%	0.52	0.49	0.51	0.52	0.114
K-anonymity (1000)	K-anonymity (1000)	0.1%	0.1%	0.12	0.12	0.11	0.08	0.548
Original graph		100%	100%	0.86	0.85	0.85	0.83	0.000

5.2.2. SCALABILITY.

Figure 4(c) shows scalability of each graph generative model. VGAE and GraphAF meet out-of-memory errors on MS Physic and MS CS, respectively. GraphDF takes more than 20 hours on the third smallest dataset, AmazonP. As GraphDF does not generate any meaningful graph structures even on the Cora and Citeseer datasets, we stop running GraphDF and declare an out-of-time error. These results are not surprising, given they are originally designed for small-size molecule graphs, thus having many un-parallelizable operations. Only CGT and GraphEBM scale to all graphs successfully. However, note that GraphEBM fails to learn any meaningful distributions from the original graphs as shown in Figures 4(a) and 4(b). In Appendix A.5, we show our proposed CGT scales to ogbn-arxiv ($170K$ nodes and $1.2M$ edges) and ogbn-products ($2.4M$ nodes and $61.8M$ edges) successfully.

5.2.3. PRIVACY.

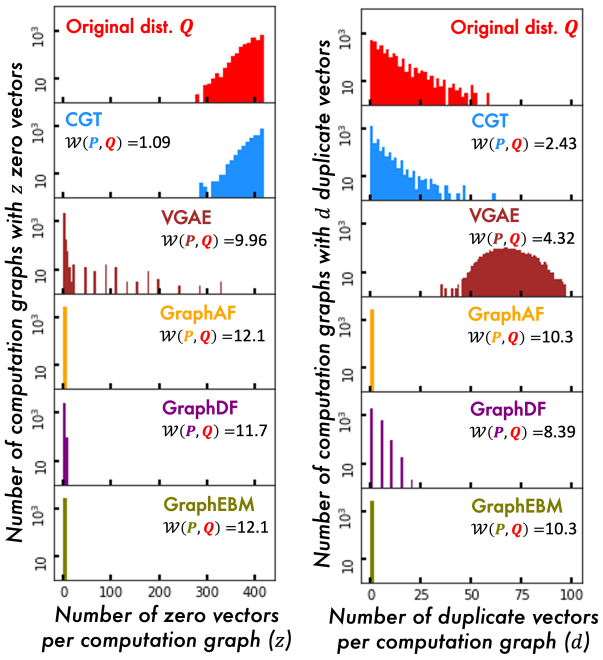
As none of our baseline generative models provides privacy guarantees, we examine the performance-privacy trade-off across different privacy guarantees on the Cora dataset only using our method. For k-anonymity, we use the k-means clustering algorithm (Bradley et al., 2000) varying the minimum cluster size k. For Differential Privacy (DP) for node attributes, we use DP k-means (Chang et al., 2021) varying the privacy cost ϵ while setting $\delta = 0.01$. In Table 1, higher k and smaller ϵ (i.e., stronger privacy) hinder the generative model's ability to learn the exact distributions of the original graphs; thus, the GNN performance gaps between original and generated graphs increase (lower Pearson and Spearman correlations). To provide DP for edge distributions, we use DP stochastic gradient descent (Song et al., 2013) to train the transformer, varying the privacy cost ϵ while setting $\delta = 0.1$. In Table 1, even with astronomically low privacy cost ($\epsilon = 10^6$), the performance of our generative model degrades significantly. When we set $\epsilon = 10^9$ (which is impractical), we can finally see a reasonable performance. This shows the limited performance of DP SGD on the transformer architecture. Detailed GNN accuracies could be found in Appendix A.7.

To verify the effectiveness of K-anonymity in terms of re-identification attacks, we compare it with simple privacy baselines that add noise on nodes/edges as follow:

- **Edge addition:** We add x times more random edges than the original number of edges. Given a corrupted graph, an original edge can be re-identified with a probability of $1/x$.
- **Edge deletion:** We delete $x\%$ of edges from the original graph. Given a corrupted graph, an original edge can be re-identified with a probability of $(100 - x)/100\%$.
- **Noise addition to node attributes:** Given a binary node attribute vector, when s elements in the vector are '1', we randomly flip '0' to '1' for xs times. Given a corrupted graph, an original attribute can be re-identified with a probability of $1/x$.
- **K-anonymity:** As described in the paper, given a corrupted graph, a node attribute vector and an edge distribution of a node can be re-identified with a probability of $1/K$ (Claim 1 in the original paper).

We run four GNN models (GCN, SGC, GIN, GAT) with different privacy approaches on the Cora dataset and computed MSE between GNN performance on the original and synthetic (corrupted) graphs. As presented in the table, K-anonymity (K=5) shows the smallest MSE (0.001) while providing stronger privacy guarantees (20% re-identification for both node and edge distribution) than the baselines of adding noise. For instance, the edge deletion (50%, 3rd row) also shows the smallest MSE (0.001), but this approach does not guarantee any privacy for node attributes and provides a 50% chance of successful edge re-identification. Note that K-anonymity ($K = 100$), which provides a 1% re-identification ratio, shows lower MSE (0.010) than most of the other baselines.

These results are not surprising, according to a recent work (Epasto et al., 2022) that analyzes noise required for privacy guarantees on graph data. (Epasto et al., 2022)



(a) (b)

Figure 5: **CGT preserves distributions of graph statistics in generated graphs:** Duplicate encoding encodes graph structure into feature matrices of computation graphs. In each computation graph, # zero vectors is inversely proportional to node degree, while # redundant vectors is proportional to edge density. We measure Wasserstein distance $\mathcal{W}(P, Q)$ between the original distribution Q and the distribution P generated by each baseline.

shows that the noise addition approach does not work well for low-degree nodes and requires many mutations to provide strong privacy guarantees. However, as we stated in the limitations of this work (Appendix A.2), we need stronger privacy guarantees than K-anonymity to use the generator in practice. We believe that by formally defining the benchmark graph generation problem and providing an end-to-end framework where we can easily adapt off-the-shelf state-of-the-art privacy modules (e.g., differential privacy), we can promote more research in this direction.

5.3. Graph statistics.

Given a source graph, our method generates a set of computation graphs without any node ids. In other words, attackers cannot merge the generated computation graphs to restore the original graph and re-identify node information. Thus, instead of traditional graph statistics such as orbit counts or clustering coefficients that rely on the global view of graphs, we define new graph statistics for computation graphs that are encoded by the duplicate scheme.

Duplicate scheme fixes adjacency matrices across all computation graphs by infusing structural information (originally encoded in adjacency matrices) into feature matrices.

- **Number of zero vectors:** In duplicate-encoded feature matrices, zero vectors correspond to null nodes that are padded when a node has fewer neighbors than a sampling neighbor number. This metric is inversely proportional to *node degree distributions* of the underlying graph.
- **Number of duplicate feature vectors:** Feature vectors are duplicated when nodes share neighbors. This metric is proportional to number of cycles in a computation graph, indicating the *edge density* of the underlying graph.

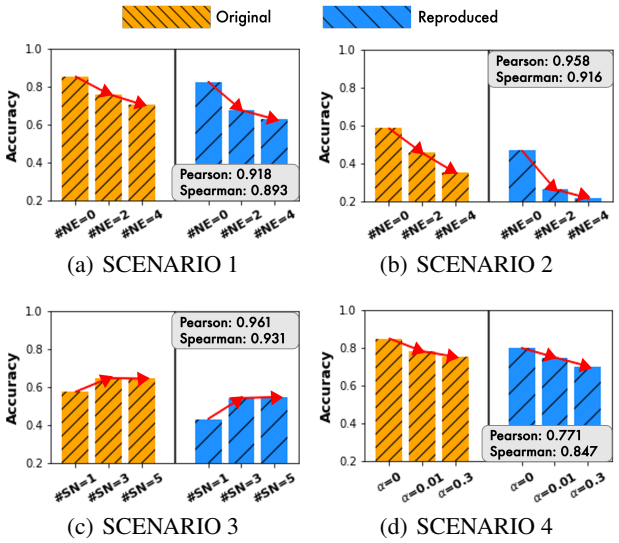
For fair comparison, we provide the same set of duplicate-encoded computation graphs to each baseline as CGT, then compute the two proxy graph statistics we described above in each generated computation graph. In Figure 5, we plot the distributions of this two statistics generated by each baseline. Only our method successfully preserves the distributions of the graph statistics on the generated computation graphs with up to 11.01 smaller Wasserstein distance than other baselines.

In Figure 5(a), the competing baselines have basically no zero vectors in the computation graphs. In the set of duplicate-encoded computation graphs given to each baseline, the input graph structures are fixed with variable feature matrices. GraphAF, GraphDF, and GraphEBM all fail to learn the distributions of feature vectors (i.e., the number of zero vectors in each computation graph) and generate highly dense feature matrices for almost all computation graphs. This shows that the existing graph generative models cannot jointly learn the distribution of node features with graph structures.

5.4. Various scenarios to evaluate benchmark effectiveness

To study the benchmark effectiveness of our generative model in depth, we design 4 different scenarios where GNN performance varies widely. In each scenario, we make 3 variations of an original graph and evaluate whether our graph generative model can reproduce these variations. In Figure 6, we report average performance of 4 GNN models on each variation. *We expect the performance trends across variations of the original graph to be reproduced across variations of synthetic graphs.* Due to the space limitation, we present results on the AmazonP dataset in Figure 6. Other datasets with detailed GNN accuracies can be found in Appendix A.9.

SCENARIO 1: noisy edges on aggregation strategies. We choose 4 GNN models with different aggregation strategies: GCN with mean aggregator, GIN with sum aggregator, SGC with linear aggregator, and GAT with attention aggregator. We make 3 variations of the original graph by adding different numbers of noisy edges ($\#NE$) to each node. In Figure 6(a), when more noisy edges are added, the GNN



(a) SCENARIO 1 (b) SCENARIO 2

(c) SCENARIO 3 (d) SCENARIO 4

Figure 6: CGT reproduces GNN performance changes with different number of noisy edges ($\#NE$), sampled neighbors ($\#SN$), and different amount of distribution shifts (α) successfully.

accuracy drops in the original graph. These trends are exactly reproduced on the generated graph with 0.918 Pearson correlation, showing our method successfully reproduces different amount of noisy edges in the original graphs.

SCENARIO 2: noisy edges on neighbor sampling. We choose 4 GNN models with different neighbor sampling strategies: GraphSage with random sampling, FastGCN with heuristic layer-wise sampling, AS-GCN with trainable layer-wise sampling, and PASS with trainable node-wise sampling. We make 3 variations of the original graph by adding noisy edges ($\#NE$) as in SCENARIO 1. In Figure 6(b), when more noisy edges are added, the sampling accuracy drops in the original graph. This trend is reproduced in the generated graph, showing 0.958 Pearson correlation.

SCENARIO 3: different sampling numbers on neighbor sampling. We choose the same 4 GNN models with different neighbor sampling strategies as in SCENARIO 2. We make 3 variations of the original graph by changing the number of sampled neighbor nodes ($\#SN$). As shown in Figure 6(c), trends among original graphs — GNN performance increases sharply from $\#SN = 1$ to $\#SN = 3$, then slowly from $\#SN = 3$ to $\#SN = 5$— are successfully captured in the generated graphs with up to 0.961 Pearson correlation. This shows CGT reproduces the neighbor distributions successfully.

SCENARIO 4: distribution shift. (Zhu et al., 2021) proposed a biased training set sampler to examine each GNN model's robustness to distribution shift between the training/test time. The biased sampler picks a few seed nodes and finds nearby nodes using the Personalized PageRank vectors (Page et al., 1999) $\pi_{\text{ppr}} = (I - (1-\alpha)\tilde{A})^{-1}$ with decaying coefficient α, then uses them to compose a biased training set. The higher α is, the larger the distribution is shifted between training/test sets. We make 3 variations of the original graph by varying α and check how 4 different GNN models, GCN, SGC, GAT, and PPNP, deal with the biased training set. In Figure 6(d), the performance of GNN models drops as α increases on the original graphs. This trend is reproduced on generated graphs, showing that CGT can capture train/test distribution shifts successfully.

Table 3: **Ablation study**

Model	MSE (↓)	Pearson (↑)	Spearman (↑)
w/o Label	0.067	0.592	0.591
w/o Position	0.072	0.411	0.413
w/o Attention	0.085	0.329	0.286
w/o All	0.034	0.739	0.574
CGT (Ours)	0.017	0.943	0.914

5.5. Ablation study

To show the importance of each component in our proposed model, we run four ablation studies: CGT without 1) label conditioning, 2) position embedding trick, 3) masked attention trick, and 4) all three modules (i.e., original Transformer) We run 9 GNN models on 3 datasets (Cora, Citeseer, Pubmed) and compare the 9×3 pairs of GNN accuracies on original and generated graphs. When we remove the position embedding trick, we provide the different position embeddings to all nodes in a computation graph, following the original transformer architecture. When we remove attention masks from our model, the transformer attends all other nodes in the computation graphs to compute the context embeddings. As shown in Table 3, removing any component negatively impacts the model performance.

6. Conclusion

We propose a new graph generative model CGT that (1) generates effective benchmark graphs on which GNNs show similar performance as on the source graphs, (2) scales to process large-scale graphs, and (3) incorporates off-the-shelf privacy modules to guarantee end-user privacy of the generated graph. We hope our work sparks further research to address the limited access to (highly proprietary) real-world graphs, enabling the community to develop new GNN models on challenging, realistic problems.

7. Acknowledgement

We thank Alessandro Epasto for discussions on related work. MY gratefully acknowledges support from Amazon Graduate Research Fellowship. GPUs are partially supported by AWS Cloud Credit for Research program.

References

Albert, R. and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Benamira, A., Devillers, B., Lesot, E., Ray, A. K., Saadi, M., and Malliaros, F. D. Semi-supervised learning and graph neural networks for fake news detection. In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 568–569. IEEE, 2019.

Bradley, P. S., Bennett, K. P., and Demiriz, A. Constrained k-means clustering. *Microsoft Research, Redmond*, 20(0): 0, 2000.

Chakrabarti, D. and Faloutsos, C. Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)*, 38(1):2–es, 2006.

Chami, I., Abu-El-Haija, S., Perozzi, B., RÃ©, C., and Murphy, K. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022.

Chang, A., Ghazi, B., Kumar, R., and Manurangsi, P. Locally private k-means in one round. In *International Conference on Machine Learning*, pp. 1441–1451. PMLR, 2021.

Chen, J., Ma, T., and Xiao, C. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

De Cao, N. and Kipf, T. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Dwork, C. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pp. 1–19. Springer, 2008.

Epasto, A., Mirrokni, V., Perozzi, B., Tsitsulin, A., and Zhong, P. Differentially private graph learning via sensitivity-bounded personalized pagerank. *arXiv preprint arXiv:2207.06944*, 2022.

Erdős, P., Rényi, A., et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Eswaran, D., Rabbany, R., Dubrawski, A. W., and Faloutsos, C. Social-affiliation networks: Patterns and the soar model. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 105–121. Springer, 2018.

Fey, M., Lenssen, J. E., Weichert, F., and Leskovec, J. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning*, pp. 3294–3304. PMLR, 2021.

Friedman, A. and Schuster, A. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 493–502, 2010.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Grover, A., Zweig, A., and Ermon, S. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*, pp. 2434–2444. PMLR, 2019.

Guo, X. and Zhao, L. A systematic survey on deep generative models for graph generation. *arXiv preprint arXiv:2007.06686*, 2020.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Huang, W., Zhang, T., Rong, Y., and Huang, J. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems*, 31, 2018.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pp. 2323–2332. PMLR, 2018.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Leskovec, J. and Faloutsos, C. Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*, pp. 497–504, 2007.

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. Kronecker graphs: an approach to modeling networks. *Journal of Machine Learning Research*, 11(2), 2010.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. Efficient graph generation with graph recurrent attention networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Liu, M., Luo, Y., Wang, L., Xie, Y., Yuan, H., Gui, S., Yu, H., Xu, Z., Zhang, J., Liu, Y., et al. Dig: a turnkey library for diving into graph deep learning research. *Journal of Machine Learning Research*, 22(240):1–9, 2021.

Liu, Z., Wu, Z., Zhang, Z., Zhou, J., Yang, S., Song, L., and Qi, Y. Bandit samplers for training graph neural networks. *arXiv preprint arXiv:2006.05806*, 2020.

Luo, Y., Yan, K., and Ji, S. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pp. 7192–7203. PMLR, 2021.

Myers, J. L., Well, A. D., and Lorch Jr, R. F. *Research design and statistical analysis*. Routledge, 2013.

Narayanan, S. D., Sinha, A., Jain, P., Kar, P., and Sellamanickam, S. Iglu: Efficient gcn training via lazy updates. *arXiv preprint arXiv:2109.13995*, 2021.

Olatunji, I. E., Funke, T., and Khosla, M. Releasing graph neural networks with differential privacy guarantees. *arXiv preprint arXiv:2109.08907*, 2021.

Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Palowitch, J., Tsitsulin, A., Mayer, B., and Perozzi, B. Graphworld: Fake graphs bring real insights for gnns. *arXiv preprint arXiv:2203.00112*, 2022.

Popova, M., Shvets, M., Oliva, J., and Isayev, O. Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019.

Proserpio, D., Goldberg, S., and McSherry, F. A workflow for differentially-private graph synthesis. In *Proceedings of the 2012 ACM workshop on Workshop on online social networks*, pp. 13–18, 2012.

Qin, Z., Yu, T., Yang, Y., Khalil, I., Xiao, X., and Ren, K. Generating synthetic decentralized social graphs with local differential privacy. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 425–438, 2017.

Sajadmanesh, S. and Gatica-Perez, D. Locally private graph neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2130–2145, 2021.

Sala, A., Zhao, X., Wilson, C., Zheng, H., and Zhao, B. Y. Sharing graphs using differentially private graph models. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pp. 81–98, 2011.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pp. 412–422. Springer, 2018.

Song, S., Chaudhuri, K., and Sarwate, A. D. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pp. 245–248. IEEE, 2013.

Suhail, M., Mittal, A., Siddiquie, B., Broaddus, C., Eledath, J., Medioni, G., and Sigal, L. Energy-based learning for

scene graph generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13936–13945, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, D., Lin, J., Cui, P., Jia, Q., Wang, Z., Fang, Y., Yu, Q., Zhou, J., Yang, S., and Qi, Y. A semi-supervised graph attentive network for financial fraud detection. In *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 598–607. IEEE, 2019.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Xiao, Q., Chen, R., and Tan, K.-L. Differentially private network data release via structural inference. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 911–920, 2014.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Yang, C., Wang, H., Zhang, K., Chen, L., and Sun, L. Secure deep graph generation with link differential privacy. *arXiv preprint arXiv:2005.00455*, 2020.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

Yoon, M., Gervet, T., Shi, B., Niu, S., He, Q., and Yang, J. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2046–2056, 2021.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pp. 5708–5717. PMLR, 2018.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Malevich, A., Kannan, R., Prasanna, V., Jin, L., and Chen, R. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems*, 34: 19665–19679, 2021.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., and Li, H. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2019.

Zhu, Q., Ponomareva, N., Han, J., and Perozzi, B. Shift-robust gnns: Overcoming the limitations of localized graph training data. *Advances in Neural Information Processing Systems*, 34, 2021.

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *NIPS*, pp. 11249–11259, 2019.

A. Appendix

A.1. Reproducibility

Our code is publicly available [1]. Dataset information can be found in Appendix A.15 and can be downloaded from the open data source [2]. Open source libraries for DP K-means and DP-SGD we used are listed in Appendix A.13. Baseline graph generative models and their open source libraries are described in Appendix A.15. GNN models we benchmark during experiments and their open source libraries are described in Appendix A.11.1.

A.2. Limitation of the study

This paper shows that clustering-based solutions can achieve k-anonymity privacy guarantees. We stress, however, that implementing a real-world system with strong privacy guarantees will need to consider many other aspects beyond the scope of this paper. We leave as future work the study of whether we can combine stronger privacy guarantees with those of k-anonymity to enhance privacy protection

A.3. Computation graph sampling in GNN training

The main challenge of adapting GNNs to large-scale graphs is that GNNs expand neighbors recursively in the aggregation operations, leading to high computation and memory footprints. For instance, if the graph is dense or has many high degree nodes, GNNs need to aggregate a huge number of neighbors for most of the training/test examples. To alleviate this neighbor explosion problem, GraphSage (Hamilton et al., 2017) proposed to sample a fixed number of neighbors in the aggregation operation, thereby regulating the computation time and memory usage.

To train a L-layered GNN model with a user-specified neighbor sampling number s, a computation graph is generated for each node in a top-down manner ($l : L \rightarrow 1$): A target node v is located at the L-th layer; the target node samples s neighbors, and the sampled s nodes are located at the $(L-1)$-th layer; each node samples s neighbors, and the sampled s^2 nodes are located at the $(L-2)$-th layer; repeat until the 1-st layer. When the neighborhood is smaller than s, we sample all existing neighbors of the node. Which nodes to sample varies across different sampling algorithms. The sampling algorithms for GNNs broadly fall into two categories: node-wise sampling and layer-wise sampling.

- **Node-Wise Sampling.** The sampling distribution $q(j|i)$ is defined as a probability of sampling node v_j given a source node v_i. In node-wise sampling, each node samples k neighbors from its sampling distribution, then the total number of nodes in the l-th layer becomes $O(k^l)$. GraphSage (Hamilton et al., 2017) is one of the most well-known node-wise sampling method with the uniform sampling distribution $q(j|i) = \frac{1}{N(i)}$. GCN-BS (Liu et al., 2020) introduces a variance reduced sampler based on multi-armed bandits, and PASS (Yoon et al., 2021) proposes a performance-adaptive node-wise sampler.

- **Layer-Wise Sampling.** To alleviate the exponential neighbor expansion $O(k^l)$ of the node-wise samplers, layer-wise samplers define the sampling distribution $q(j|i_1, \cdots, i_n)$ as a probability of sampling node v_j given a set of nodes $\{v_k\}_{k=i_1}^{i_n}$ in the previous layer. Each layer samples k neighbors from their sampling distribution $q(j|i_1, \cdots, i_n)$, then the number of sampled nodes in each layer becomes $O(k)$. FastGCN (Chen et al., 2018) defines $q(j|i_1, \cdots, i_n)$ proportional to the degree of the target node v_j, thus every layer has independent-identical-distributions. LADIES (Zou et al., 2019) adopts the same iid as FastGCN but limits the sampling domain to the neighborhood of the sampler layer. AS-GCN (Huang et al., 2018) parameterizes the sampling distributions $q(j|i_1, i_2, \ldots, i_n)$ with a learnable linear function. While the layer-wise samplers successfully regulate the neighbor expansion, they suffer from sparse connection problems — some nodes fail to sample any neighbors while other nodes sample their neighbors repeatedly in a given layer.

Note that the layer-wise samplers also define a maximum number of neighbors to sample (but per each layer) and the depth of computation graphs as the depth of the GNN model. All sampling methods we describe above can be applied to our computation graph sampling module described in Section 3.2. As the depth of computation graph L is decided by the depth of GNN models, oversmoothing (Li et al., 2018) or oversquashing (Alon & Yahav, 2020) could happen with the deep GNN models. To handle this issue, (Zeng et al., 2021) proposes to disentangle the depth of computation graphs and the depth of GNN models, then limit the computation graph sizes to small to avoid oversmoothing/oversquashing.

There are many different clustering or subgraph sampling methodologies other than what we described above. Note that, even after we get subgraphs using any clustering/subgraph sampling methods, to do message-passing under GCN models, each node eventually has a tree-structure-shaped computation graph that is composed of nodes engaged in the node's embedding computation. In other words, CGT receives subgraphs sampled by ClusterGCN (Chiang et al., 2019) and GraphSAINT (Zeng et al., 2019) and extracts a computation graph for each node (in this case, we can set the sampling number as the maximum degree in the subgraph not to lose any further neighbors by sampling). GNNAutoScale (Fey et al., 2021) and IGLU (Narayanan et al., 2021) are recently proposed frameworks for scaling

[1] https://github.com/minjiyoon/CGT

[2] https://github.com/shchur/gnn-benchmark

arbitrary message-passing GNNs to large graphs, as an alternative paradigm to neighbor sampling. As our method adopts neighbor sampling — the most common way to deal with the scalability issue of GNNs so far — we cannot directly apply our graph benchmark generation method to these methods. This is an interesting avenue for future work.

A.4. Proof of privacy and scalability claims

Claim 1 (k-Anonymity for node attributes and edge distributions)**.** *In the generated computation graphs, each node attribute and edge distribution appear at least k times, respectively.*

Proof. In the quantization phase, we use the k-means clustering algorithm (Bradley et al., 2000) with a minimum cluster size k. Then each node id is replaced with the id of the cluster it belongs to, reducing the original $(n \times n)$ graph into a $(m \times m)$ hypergraph where $m = n/k$ is the number of clusters. Then Computation Graph Transformer learns edge distributions among m hyper nodes (i.e., clusters) and generates a new $(m \times m)$ hypergraph. In the hypergraph, there are at most m different node attributes and m different edge distributions. During the de-quantization phase, a $(m \times m)$ hypergraph is mapped back to a $(n \times n)$ graph by letting k nodes in each cluster follow their cluster's node attributes/edge distributions as follows: k nodes in the same cluster will have the same feature vector that is the average feature vector of original nodes belonging to the cluster. When s denotes the number of sampled neighbor nodes, each node samples s clusters (with replacement) following its cluster's edge distributions among m clusters. When a node samples cluster i, it will be connected to one of nodes in the cluster i randomly. At the end, each node will have s neighbor nodes randomly sampled from s clusters the node samples with the cluster's edge distribution, respectively. Likewise, all k nodes belonging to the same cluster will sample neighbors following the same edge distributions. Thus each node attribute and edge distribution appear at least k times in a generated graph. ■

Claim 2 ((ϵ, δ)-Differential Privacy for node attributes)**.** *With probability at least $1-\delta$, our generative model A gives ϵ-differential privacy for any graph $\mathcal{G}$, any neighboring graph $\mathcal{G}_{-v}$ without any node $v \in \mathcal{G}$, and any new computation graph $\mathcal{G}_{cg}$ generated from our model as follows:*

$$e^{-\epsilon} \leq \frac{Pr[A(\mathcal{G}) = \mathcal{G}_{cg}]}{Pr[A(\mathcal{G}_{-v}) = \mathcal{G}_{cg}]} \leq e^{\epsilon}$$

Proof. $\mathcal{G}_{-v}$ denotes neighboring graphs to the original one $\mathcal{G}$, but without a specific node v. During the quantization phase, we use (ϵ, δ)-differential private k-means clustering algorithm on node features (Chang et al., 2021). Then clustering results are differentially private with regard to each node features. In the generated graphs, each node feature is decided by the clustering results (i.e., the average feature vector of nodes belonging to the same cluster). Then, by looking at the generated node features, one cannot tell whether any individual node feature was included in the original dataset or not. ■

Remark 1 ((ϵ, δ)-Differential Privacy for edge distributions)**.** In our model, individual nodes' edge distributions are learned and generated by the transformer. When we use (ϵ, δ)-differential private stochastic gradient descent (DP-SGD) (Song et al., 2013) to train the transformer, the transformer becomes differentially private in the sense that by looking at the output (generated edge distributions), one cannot tell whether any individual node's edge distribution (input to the transformer) was included in the original dataset or not. If we have DP-SGD that can train transformers successfully with reasonably small ϵ and δ, we can guarantee (ϵ, δ)-differential privacy for edge distribution of any graph generated by our generative model. However, as we show in Section 5.2.3, current DP-SGD is not stable yet for transformer training, leading to very coarse or impractical privacy guarantees.

Claim 3 (Scalability)**.** *When we aim to generate L-layered computation graphs with neighbor sampling number s on a graph with n nodes, computational complexity of* CGT *training is $O(s^{2L}n)$, and that of the cost-efficient version is $O(L^2 s^L n)$.*

Proof. During k-means, we randomly sample n_k node features to compute the cluster centers. Then we map each feature vector to the closest cluster center. By sampling n_k nodes, we limit the k-mean computation cost to $O(n_k^2)$. The sequence flattened from each computation graph is $O(1 + s + \cdots + s^L)$ and the number of sequences (computation graphs) is $O(n)$. Then the training time of the transformer is proportional to $O(s^{2L}n)$. In total, the complexity is $O(s^{2L}n + n_k^2)$. As $s^{2L}n >> n_k^2$, the final computation complexity becomes $O(s^{2L}n)$. In the cost-efficient version, the length of sequences (composed only of direct ancestor nodes) is reduced to L. However, the number of sequences increases to $s^L n$ because each nodes has one computation graph composed of s^L shortened sequences. Then the final computation complexity become $O(L^2 s^L n)$. ■

A.5. CGT on ogbn-arxiv and ogbn-products

To examine its scalability, we run CGT on two large-scale datasets, ogbn-arxiv and ogbn-products (Hu et al., 2020). We run CGT on 4 NVIDIA TITAN X GPUs with 12 GB memory size with sampling number 5 and $K = 30$ for K-anonymity. In Table 4, CGT takes 1.1 hours for ogbn-arxiv with $170K$ nodes and $1.2M$ edges, while taking 14.7 hours for ogbn-products with $2.4M$ nodes and $61.8M$ edges. This

Table 4: **CGT on ogbn-arxiv and ogbn-products:** *Training time (hr)* column denotes the total training/generation time of CGT.

Dataset	Node num	Edge num	Noise num	Model	Original acc.	Generated acc/	MSE	Training time (hr)	Pearson
ogbn-arxiv	169,343	1,166,243	0	GCN	0.69	0.7	0.00032	1.1	0.989
				SGC	0.68	0.7			
				GIN	0.69	0.71			
				GAT	0.69	0.71			
			2	GCN	0.58	0.6	0.00015	1.7	
				SGC	0.57	0.58			
				GIN	0.61	0.62			
				GAT	0.62	0.62			
			4	GCN	0.53	0.55	0.00015	2.8	
				SGC	0.54	0.53			
				GIN	0.56	0.56			
				GAT	0.57	0.58			
ogbn-products	2,449,029	61,859,140	0	GCN	0.87	0.89	0.00258	14.7	
				SGC	0.75	0.84			
				GIN	0.86	0.89			
				GAT	0.87	0.9			

Table 5: **CGT as training/test set generators:** We replace the original training/test sets of the target dataset (Cora) with irrelevant graphs (Citeseer or Pubmed) and synthetic Cora generated by our proposed CGT.

Train set	Test set	Accuracy
Cora	Cora	0.86
Citeseer	Cora	0.14
Pubmed	Cora	0.09
Synthetic Cora (CGT)	Cora	0.77
Cora	Synthetic Cora (CGT)	0.74
Synthetic Cora (CGT)	Synthetic Cora (CGT)	0.76

shows CGT's strong scalability. In terms of benchmark effectiveness, CGT shows low MSE (up to 1.5×10^{-4}) and high Pearson correlation (0.989). Note that we could not compare with other baselines as they all fail to scale even on MS Physic dataset with with $35K$ nodes and $248K$ edges (Figure 4(c)).

A.6. CGT as training/test set generators

In this experiment, we train GNNs on synthetic graphs generated by CGT and test them on real graphs, and vice versa. For comparison, we train GCN on the two independent graphs (Citeseer and Pubmed) and test on the target graph (Cora). Since the feature dimensions of Citeseer and Pubmed differ from those of Cora, we mapped the original node feature vectors to Cora's feature dimension using PCA. The results in the Table 5 demonstrate that our CGT generates synthetic graphs that follow Cora's distribution and preserve high accuracy, whereas GCN models trained on Citeseer and Pubmed show low accuracy on Cora. The accuracy drop induced by CGT is mainly due to privacy, as we provided 30-Anonymity in this experiment. We conducted a similar experiment in Section 5.4 SCENARIO 4, where we prepared different distributions for the training and test sets of GNNs. As the distribution shift becomes larger, the performance of GNNs drops. Our proposed CGT successfully reproduces this distribution shift, and thus, it also reproduces the performance drop in the generated graphs.

A.7. Detailed GNN performance in the privacy experiment in Section 5.2.3

Table 6 shows detailed privacy-GNN performance trade-off on the Cora dataset. In k-anonymity, higher k (i.e., more nodes in the same clusters, thus stronger privacy) hinders the generative model's ability to learn the exact distributions of the original graphs, and the GNN performance gaps between original and generated graphs increase, showing lower Pearson and Spearman coefficients. DP kmeans shows higher Pearson and Spearman coefficients with smaller ϵ values (i.e., stronger privacy). However, when we examine the detailed GNN performance, we observe that GNN accuracy is significantly lower with smaller ϵ values. For your convenience, we compare their MSE from the original accuracy as well as the correlation coefficients in Table 6: MSE is descreasing from $0.134(\epsilon = 1)$ to $0.093(\epsilon = 10)$ and $0.063(\epsilon = 25)$. Stronger privacy can lead to higher correlations as DP k-means can remove noise in graphs (while hiding outliers for privacy) and capture representative distributions from the original graph more effectively. While DP kmeans is capable of providing reasonable privacy to node attribute distributions, DP-SGD is impractical, showing low GNN performance even with astronomically low privacy cost ($\epsilon = 10^6$) as explained in Section 4.2. Note that reasonable ϵ values typically range between 0.1 and 5.

Table 6: **Privacy-Performance trade-off in graph generation on the Cora dataset**

#NE	model	Original	No privacy	K-anonymity			DP kmean ($\delta = 0.01$)			DP SGD ($\delta = 0.1$)	
				$k = 100$	$k = 500$	$k = 1000$	$\epsilon = 1$	$\epsilon = 10$	$\epsilon = 25$	$\epsilon = 10^6$	$\epsilon = 10^9$
0	**GCN**	0.860	0.760	0.750	0.520	0.120	0.530	0.570	0.650	0.130	0.640
	SGC	0.850	0.750	0.740	0.490	0.120	0.510	0.590	0.620	0.150	0.620
	GIN	0.850	0.750	0.760	0.510	0.110	0.520	0.570	0.650	0.140	0.640
	GAT	0.830	0.750	0.740	0.520	0.080	0.440	0.560	0.640	0.140	0.610
2	**GCN**	0.770	0.680	0.570	0.380	0.110	0.500	0.400	0.450	0.110	0.580
	SGC	0.770	0.680	0.580	0.360	0.080	0.350	0.410	0.450	0.140	0.570
	GIN	0.780	0.670	0.590	0.390	0.140	0.390	0.410	0.470	0.140	0.580
	GAT	0.680	0.660	0.560	0.380	0.110	0.350	0.390	0.430	0.120	0.530
4	**GCN**	0.720	0.610	0.510	0.280	0.090	0.280	0.390	0.430	0.100	0.410
	SGC	0.720	0.600	0.500	0.280	0.110	0.300	0.410	0.450	0.140	0.410
	GIN	0.660	0.590	0.480	0.300	0.160	0.320	0.410	0.460	0.150	0.400
	GAT	0.600	0.570	0.470	0.290	0.080	0.250	0.370	0.450	0.140	0.380
Pearson		1.000	0.934	0.916	0.862	0.030	0.874	0.844	0.804	0.112	0.890
Spearman		1.000	0.935	0.947	0.812	0.018	0.869	0.805	0.807	0.116	0.959
MSE		0.000	0.008	0.026	0.136	0.427	0.134	0.093	0.063	0.396	0.053

A.8. Additional experiments on graph statistics

Figure 7 shows distributions of graph statistics on computation graphs sampled from the original/quantized/generated graphs. Quantized graphs are graphs after the quantization process: each feature vector is replaced by the mean feature vector of a cluster it belongs to, and adjacency matrices are a constant encoded by the duplicate encoding scheme. Quantized graphs are input to CGT, and generated graphs are output from CGT as presented in Figure 2. While converting from original graphs to quantized graphs, CGT trades off some of the graph statistics information for k-anonymity privacy benefits. In Figure 7, we can see distributions of graphs statistics have changed slightly from original graphs to quantized graphs. Then CGT learns distributions of graph statistics on the quantized graphs and generates synthetic graphs. The variations given by CGT are presented as differences in distributions between quantized and generated graphs in Figure 7.

A.9. Detailed GNN performance in the benchmark effectiveness experiment in Section 5.4

Tables 7, 8, 9, and 10 show GNN performance on node classification tasks across the original/quantized/generated graphs. Quantized graphs are graphs after the quantization process: each feature vector is replaced by the mean feature vector of a cluster it belongs to, and adjacency matrices are a constant encoded by the duplicate encoding scheme. Quantized graphs are input to CGT, and generated graphs are output from CGT as presented in Figure 2. As presented across all four tables, our proposed generative model CGT successfully generates synthetic substitutes of large-scale real-world graphs that shows similar task performance as on the original graphs.

Link prediction. As nodes are the minimum unit in graphs that compose edges or subgraphs, we can generate subgraphs for edges by merging computation graphs of their component nodes. Here we show link prediction results on original graphs are also preserved successfully on our generated graphs. We run GCN, SGC, GIN, and GAT on graphs, followed by Dot product or MLP to predict link probabilities. Table 11 shows Pearson and Spearman correlations across 8 different combinations of link prediction models (4 GNN models $\times$ 2 link predictors) on each dataset and across the whole datasets. Our model generates graphs that substitute original graphs successfully, preserving the ranking of GNN link prediction performance with 0.754 Spearman correlation across the datasets.

A.10. Detailed GNN performance in the ablation study in Section 5.5

Table 12 shows CGT without label conditioning (conditioning on the label of the root node of the computation graph), positional embedding trick (giving the same positional embedding to nodes at the same layers on the computation graph), masked attention trick (attended only on direct ancestor nodes on the computation graph), and all modules (pure Transformer) respectively. Note that this experiment is done on the original version of CGT (not the cost-efficient version in Figure 3(c)). When we remove the positional embedding trick, we provide the different positional embeddings to all nodes in a computation graph, following the original transformer architecture. When we remove attention masks from our model, the transformer attends all other nodes in the computation graphs to compute the context embeddings.

A.11. Graph Neural Networks

We briefly review graph neural networks (GNNs) then describe how neighbor sampling operations can be applied on GNNs.

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with n nodes $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$. Denote an adjacency matrix $A = (a(v_i, v_j)) \in \mathbb{R}^{n \times n}$ and a feature matrix $X \in \mathbb{R}^{n \times d}$

Table 7: **GNN performance on SCENARIO 1: noisy edges on aggregation strategies.**

Dataset	#NE	model	Original	std	Cluster	std	Generated	std	pearson	spearman
Cora	0	GCN	0.860	0.002	0.830	0.002	0.760	0.005	0.934	0.950
		SGC	0.850	0.001	0.820	0.004	0.750	0.002		
		GIN	0.850	0.004	0.830	0.008	0.750	0.013		
		GAT	0.830	0.002	0.830	0.002	0.750	0.006		
	2	GCN	0.770	0.008	0.750	0.009	0.680	0.014		
		SGC	0.770	0.008	0.740	0.003	0.680	0.015		
		GIN	0.780	0.002	0.730	0.003	0.670	0.009		
		GAT	0.680	0.013	0.740	0.002	0.660	0.009		
	4	GCN	0.720	0.011	0.690	0.008	0.610	0.015		
		SGC	0.720	0.005	0.690	0.004	0.600	0.007		
		GIN	0.660	0.019	0.680	0.007	0.590	0.016		
		GAT	0.600	0.019	0.670	0.008	0.570	0.015		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Citeseer	0	GCN	0.730	0.004	0.680	0.002	0.590	0.024	0.991	0.964
		SGC	0.730	0.002	0.670	0.002	0.580	0.029		
		GIN	0.710	0.009	0.670	0.004	0.570	0.028		
		GAT	0.710	0.003	0.670	0.004	0.570	0.029		
	2	GCN	0.570	0.005	0.560	0.010	0.460	0.013		
		SGC	0.570	0.005	0.570	0.007	0.470	0.019		
		GIN	0.540	0.020	0.560	0.003	0.440	0.015		
		GAT	0.570	0.014	0.550	0.004	0.440	0.01		
	4	GCN	0.510	0.027	0.500	0.001	0.410	0.003		
		SGC	0.520	0.009	0.500	0.002	0.410	0.005		
		GIN	0.480	0.023	0.510	0.008	0.410	0.007		
		GAT	0.490	0.012	0.510	0.004	0.400	0.009		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Pubmed	0	GCN	0.860	0.001	0.820	0.001	0.780	0.007	0.818	0.791
		SGC	0.860	0.000	0.810	0.001	0.780	0.003		
		GIN	0.830	0.006	0.810	0.001	0.770	0.002		
		GAT	0.860	0.002	0.820	0.003	0.780	0.005		
	2	GCN	0.780	0.004	0.760	0.004	0.680	0.003		
		SGC	0.760	0.004	0.750	0.006	0.670	0.004		
		GIN	0.790	0.012	0.740	0.014	0.670	0.007		
		GAT	0.710	0.011	0.770	0.003	0.680	0.005		
	4	GCN	0.730	0.003	0.710	0.003	0.640	0.007		
		SGC	0.670	0.003	0.700	0.003	0.630	0.009		
		GIN	0.770	0.011	0.700	0.008	0.600	0.017		
		GAT	0.650	0.005	0.740	0.001	0.640	0.004		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Computer	0	GCN	0.860	0.002	0.840	0.009	0.840	0.001	0.825	0.778
		SGC	0.860	0.005	0.810	0.009	0.830	0.007		
		GIN	0.850	0.002	0.810	0.015	0.800	0.013		
		GAT	0.840	0.008	0.840	0.011	0.830	0.01		
	2	GCN	0.780	0.004	0.760	0.004	0.680	0.003		
		SGC	0.760	0.004	0.750	0.006	0.670	0.004		
		GIN	0.790	0.012	0.740	0.014	0.670	0.007		
		GAT	0.710	0.011	0.770	0.003	0.680	0.005		
	4	GCN	0.730	0.003	0.710	0.003	0.640	0.007		
		SGC	0.670	0.003	0.700	0.003	0.630	0.009		
		GIN	0.770	0.011	0.700	0.008	0.600	0.017		
		GAT	0.650	0.005	0.740	0.001	0.640	0.004		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Photo	0	GCN	0.910	0.001	0.890	0.003	0.900	0.005	0.918	0.893
		SGC	0.910	0.000	0.890	0.005	0.900	0.006		
		GIN	0.900	0.003	0.880	0.005	0.900	0.001		
		GAT	0.900	0.009	0.880	0.010	0.890	0.007		
	2	GCN	0.870	0.007	0.870	0.003	0.790	0.007		
		SGC	0.870	0.005	0.870	0.008	0.790	0.005		
		GIN	0.870	0.006	0.870	0.004	0.770	0.012		
		GAT	0.860	0.006	0.860	0.005	0.780	0.003		
	4	GCN	0.820	0.019	0.810	0.003	0.740	0.002		
		SGC	0.830	0.001	0.810	0.022	0.730	0.012		
		GIN	0.840	0.006	0.830	0.009	0.710	0.024		
		GAT	0.860	0.010	0.820	0.029	0.720	0.01		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS CS	0	GCN	0.880	0.004	0.890	0.003	0.830	0.008	0.916	0.922
		SGC	0.880	0.003	0.880	0.002	0.830	0.008		
		GIN	0.870	0.001	0.870	0.004	0.820	0.013		
		GAT	0.880	0.003	0.890	0.004	0.830	0.006		
	2	GCN	0.860	0.005	0.870	0.005	0.760	0.005		
		SGC	0.860	0.006	0.860	0.004	0.750	0.006		
		GIN	0.850	0.010	0.840	0.005	0.720	0.002		
		GAT	0.860	0.007	0.860	0.005	0.750	0.01		
	4	GCN	0.840	0.003	0.840	0.004	0.710	0.009		
		SGC	0.840	0.002	0.840	0.002	0.700	0.005		
		GIN	0.820	0.009	0.790	0.010	0.670	0.011		
		GAT	0.860	0.011	0.850	0.004	0.700	0.005		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS Physic	0	GCN	0.930	0.002	0.930	0.002	0.840	0.008	0.661	0.685
		SGC	0.920	0.001	0.920	0.001	0.840	0.007		
		GIN	0.930	0.002	0.920	0.002	0.820	0.011		
		GAT	0.930	0.005	0.930	0.000	0.840	0.007		
	2	GCN	0.910	0.000	0.910	0.001	0.770	0.004		
		SGC	0.890	0.002	0.900	0.000	0.760	0.004		
		GIN	0.910	0.009	0.900	0.002	0.750	0.008		
		GAT	0.930	0.003	0.900	0.003	0.770	0.003		
	4	GCN	0.880	0.006	0.890	0.002	0.710	0.006		
		SGC	0.860	0.003	0.880	0.002	0.710	0.005		
		GIN	0.900	0.006	0.880	0.005	0.700	0.007		
		GAT	0.930	0.002	0.890	0.001	0.720	0.004		

Table 8: **GNN performance on SCENARIO 2: noisy edges on neighbor sampling.**

Dataset	#NE	model	Original	std	Cluster	std	Generated	std	pearson	spearman
Cora	0	GraphSage	0.740	0.012	0.560	0.008	0.490	0.011	0.943	0.894
		AS-GCN	0.130	0.014	0.110	0.013	0.130	0.013		
		FastGCN	0.440	0.006	0.390	0.005	0.370	0.006		
		PASS	0.790	0.011	0.620	0.008	0.560	0.029		
	2	GraphSage	0.360	0.012	0.300	0.014	0.270	0.004		
		AS-GCN	0.130	0.013	0.110	0.010	0.130	0.016		
		FastGCN	0.320	0.010	0.290	0.007	0.280	0.008		
		PASS	0.630	0.021	0.520	0.023	0.440	0.034		
	4	GraphSage	0.130	0.005	0.150	0.007	0.170	0.015		
		AS-GCN	0.180	0.057	0.130	0.008	0.130	0.008		
		FastGCN	0.540	0.013	0.610	0.020	0.570	0.01		
		PASS	0.560	0.008	0.520	0.003	0.400	0.016		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Citeseer	0	GraphSage	0.660	0.005	0.510	0.014	0.430	0.007	0.955	0.977
		AS-GCN	0.100	0.006	0.100	0.019	0.090	0.006		
		FastGCN	0.380	0.011	0.330	0.009	0.300	0.001		
		PASS	0.680	0.008	0.530	0.012	0.440	0.006		
	2	GraphSage	0.250	0.003	0.310	0.005	0.280	0.005		
		AS-GCN	0.090	0.006	0.080	0.006	0.090	0.01		
		FastGCN	0.240	0.007	0.260	0.008	0.230	0.003		
		PASS	0.540	0.008	0.460	0.010	0.410	0.014		
	4	GraphSage	0.190	0.008	0.240	0.005	0.250	0.012		
		AS-GCN	0.110	0.012	0.100	0.021	0.100	0.004		
		FastGCN	0.210	0.004	0.210	0.006	0.200	0.014		
		PASS	0.480	0.021	0.460	0.002	0.400	0.015		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Pubmed	0	GraphSage	0.780	0.005	0.680	0.002	0.630	0.004	0.885	0.916
		AS-GCN	0.260	0.009	0.230	0.026	0.240	0.007		
		FastGCN	0.470	0.003	0.450	0.003	0.430	0.003		
		PASS	0.850	0.007	0.730	0.001	0.680	0.007		
	2	GraphSage	0.409	0.002	0.467	0.012	0.431	0.004		
		AS-GCN	0.308	0.072	0.419	0.053	0.287	0.051		
		FastGCN	0.731	0.008	0.727	0.008	0.628	0.008		
		PASS	0.812	0.007	0.697	0.000	0.587	0.008		
	4	GraphSage	0.310	0.001	0.320	0.003	0.320	0.003		
		AS-GCN	0.310	0.031	0.330	0.035	0.360	0.021		
		FastGCN	0.660	0.002	0.650	0.002	0.550	0.012		
		PASS	0.790	0.001	0.690	0.006	0.430	0.005		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Computer	0	GraphSage	0.630	0.027	0.520	0.022	0.460	0.012	0.958	0.916
		AS-GCN	0.130	0.065	0.130	0.081	0.060	0.028		
		FastGCN	0.860	0.005	0.820	0.006	0.810	0.005		
		PASS	0.720	0.014	0.590	0.004	0.540	0.009		
	2	GraphSage	0.260	0.001	0.200	0.012	0.140	0.002		
		AS-GCN	0.190	0.063	0.040	0.002	0.050	0.012		
		FastGCN	0.750	0.005	0.710	0.001	0.640	0.004		
		PASS	0.620	0.011	0.530	0.006	0.220	0.033		
	4	GraphSage	0.120	0.004	0.100	0.007	0.070	0.004		
		AS-GCN	0.090	0.045	0.050	0.018	0.100	0.037		
		FastGCN	0.650	0.004	0.620	0.001	0.570	0.006		
		PASS	0.540	0.024	0.470	0.014	0.120	0.019		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Photo	0	GraphSage	0.750	0.009	0.670	0.017	0.530	0.028	0.958	0.916
		AS-GCN	0.140	0.016	0.080	0.025	0.120	0.02		
		FastGCN	0.920	0.004	0.900	0.003	0.870	0.002		
		PASS	0.850	0.011	0.780	0.006	0.540	0.049		
	2	GraphSage	0.400	0.012	0.370	0.007	0.360	0.009		
		AS-GCN	0.120	0.014	0.140	0.041	0.110	0.027		
		FastGCN	0.870	0.005	0.880	0.003	0.810	0.01		
		PASS	0.730	0.018	0.640	0.029	0.590	0.011		
	4	GraphSage	0.260	0.009	0.200	0.016	0.200	0.014		
		AS-GCN	0.100	0.025	0.130	0.037	0.130	0.054		
		FastGCN	0.670	0.003	0.670	0.006	0.620	0.006		
		PASS	0.640	0.017	0.600	0.005	0.500	0.017		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS CS	0	GraphSage	0.750	0.003	0.680	0.005	0.520	0.007	0.974	0.956
		AS-GCN	0.090	0.027	0.070	0.035	0.070	0.016		
		FastGCN	0.920	0.001	0.910	0.001	0.820	0.001		
		PASS	0.870	0.007	0.810	0.008	0.640	0.015		
	2	GraphSage	0.320	0.002	0.350	0.003	0.240	0.080		
		AS-GCN	0.040	0.028	0.050	0.022	0.050	0.036		
		FastGCN	0.910	0.002	0.910	0.001	0.820	0.002		
		PASS	0.810	0.005	0.750	0.003	0.660	0.004		
	4	GraphSage	0.200	0.008	0.230	0.008	0.120	0.018		
		AS-GCN	0.070	0.033	0.050	0.027	0.040	0.038		
		FastGCN	0.900	0.005	0.890	0.003	0.610	0.007		
		PASS	0.790	0.013	0.730	0.005	0.500	0.011		
Dataset	**#NE**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS Physic	0	GraphSage	0.850	0.005	0.790	0.003	0.590	0.009	0.956	0.951
		AS-GCN	0.240	0.051	0.190	0.042	0.240	0.052		
		FastGCN	0.950	0.001	0.940	0.001	0.820	0.004		
		PASS	0.920	0.000	0.860	0.003	0.670	0.006		
	2	GraphSage	0.490	0.001	0.500	0.003	0.420	0.005		
		AS-GCN	0.160	0.022	0.210	0.032	0.130	0.055		
		FastGCN	0.940	0.004	0.930	0.005	0.800	0.009		
		PASS	0.900	0.009	0.840	0.008	0.690	0.012		
	4	GraphSage	0.300	0.003	0.330	0.005	0.280	0.002		
		AS-GCN	0.340	0.005	0.090	0.052	0.080	0.039		
		FastGCN	0.930	0.002	0.920	0.003	0.780	0.001		
		PASS	0.890	0.001	0.830	0.005	0.610	0.004		

Table 9: **GNN performance on SCENARIO 3: different sampling numbers on neighbor sampling.**

Dataset	#SN	model	Original	std	Cluster	std	Generated	std	pearson	spearman
Cora	0	GraphSage	0.750	0.013	0.560	0.028	0.500	0.011	0.967	0.814
		AS-GCN	0.120	0.001	0.120	0.011	0.110	0.005		
		FastGCN	0.450	0.008	0.390	0.006	0.380	0.003		
		PASS	0.800	0.007	0.600	0.008	0.540	0.003		
	2	GraphSage	0.830	0.007	0.740	0.008	0.690	0.018		
		AS-GCN	0.130	0.009	0.130	0.013	0.130	0.014		
		FastGCN	0.750	0.008	0.660	0.011	0.660	0.001		
		PASS	0.840	0.004	0.740	0.012	0.680	0.011		
	4	GraphSage	0.850	0.001	0.810	0.004	0.600	0.005		
		AS-GCN	0.130	0.022	0.140	0.029	0.150	0.046		
		FastGCN	0.870	0.004	0.820	0.007	0.640	0.008		
		PASS	0.820	0.009	0.790	0.000	0.520	0.026		
Dataset	**#SN**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Citeseer	0	GraphSage	0.680	0.014	0.500	0.011	0.440	0.016	0.973	0.904
		AS-GCN	0.110	0.013	0.090	0.005	0.100	0.006		
		FastGCN	0.370	0.011	0.330	0.003	0.330	0.015		
		PASS	0.700	0.005	0.530	0.014	0.460	0.006		
	2	GraphSage	0.710	0.004	0.610	0.006	0.560	0.003		
		AS-GCN	0.110	0.012	0.110	0.010	0.090	0.004		
		FastGCN	0.670	0.008	0.600	0.005	0.580	0.001		
		PASS	0.710	0.003	0.610	0.007	0.560	0.007		
	4	GraphSage	0.730	0.006	0.650	0.009	0.600	0.01		
		AS-GCN	0.110	0.004	0.120	0.001	0.100	0.012		
		FastGCN	0.770	0.003	0.700	0.004	0.680	0.001		
		PASS	0.730	0.002	0.650	0.004	0.580	0.009		
Dataset	**#SN**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Pubmed	1	GraphSage	0.780	0.003	0.680	0.005	0.600	0.004	0.989	0.824
		AS-GCN	0.250	0.002	0.260	0.009	0.260	0.011		
		FastGCN	0.480	0.002	0.460	0.004	0.440	0.003		
		PASS	0.860	0.002	0.720	0.004	0.660	0.003		
	3	GraphSage	0.830	0.003	0.780	0.005	0.710	0.001		
		AS-GCN	0.240	0.012	0.240	0.015	0.250	0.013		
		FastGCN	0.750	0.004	0.710	0.001	0.660	0.006		
		PASS	0.880	0.002	0.780	0.003	0.710	0.008		
	5	GraphSage	0.850	0.001	0.800	0.001	0.740	0.002		
		AS-GCN	0.260	0.021	0.260	0.007	0.240	0.02		
		FastGCN	0.860	0.002	0.800	0.003	0.740	0.002		
		PASS	0.870	0.002	0.790	0.004	0.730	0.004		
Dataset	**#SN**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Computer	1	GraphSage	0.670	0.010	0.550	0.008	0.450	0.01	0.975	0.890
		AS-GCN	0.090	0.006	0.060	0.028	0.040	0.005		
		FastGCN	0.780	0.004	0.740	0.007	0.700	0.006		
		PASS	0.750	0.000	0.620	0.018	0.530	0.02		
	3	GraphSage	0.790	0.003	0.700	0.015	0.600	0.015		
		AS-GCN	0.110	0.025	0.040	0.014	0.120	0.06		
		FastGCN	0.870	0.001	0.840	0.006	0.800	0.011		
		PASS	0.810	0.015	0.760	0.023	0.640	0.009		
	5	GraphSage	0.770	0.008	0.720	0.004	0.680	0.005		
		AS-GCN	0.120	0.085	0.100	0.057	0.030	0.007		
		FastGCN	0.850	0.003	0.830	0.000	0.790	0.01		
		PASS	0.830	0.002	0.730	0.011	0.680	0.022		
Dataset	**#SN**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Photo	1	GraphSage	0.740	0.016	0.660	0.003	0.500	0.014	0.961	0.931
		AS-GCN	0.110	0.037	0.090	0.030	0.090	0.04		
		FastGCN	0.830	0.005	0.810	0.005	0.750	0.009		
		PASS	0.850	0.011	0.730	0.026	0.520	0.01		
	3	GraphSage	0.840	0.006	0.810	0.007	0.740	0.007		
		AS-GCN	0.140	0.026	0.140	0.019	0.130	0.038		
		FastGCN	0.930	0.005	0.910	0.002	0.890	0.002		
		PASS	0.910	0.002	0.870	0.002	0.750	0.017		
	5	GraphSage	0.910	0.010	0.890	0.002	0.780	0.009		
		AS-GCN	0.860	0.021	0.850	0.021	0.790	0.031		
		FastGCN	0.110	0.005	0.050	0.001	0.110	0.021		
		PASS	0.930	0.002	0.900	0.011	0.850	0.005		
Dataset	**#SN**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS CS	1	GraphSage	0.740	0.008	0.650	0.004	0.530	0.006	0.986	0.901
		AS-GCN	0.070	0.050	0.060	0.025	0.080	0.023		
		FastGCN	0.920	0.001	0.920	0.000	0.840	0.003		
		PASS	0.870	0.005	0.770	0.005	0.690	0.004		
	3	GraphSage	0.840	0.004	0.820	0.004	0.680	0.008		
		AS-GCN	0.090	0.051	0.090	0.035	0.070	0.018		
		FastGCN	0.930	0.001	0.920	0.002	0.810	0.01		
		PASS	0.900	0.004	0.870	0.003	0.680	0.013		
	5	GraphSage	0.870	0.003	0.850	0.003	0.750	0.011		
		AS-GCN	0.060	0.044	0.040	0.002	0.110	0.037		
		FastGCN	0.930	0.001	0.920	0.000	0.810	0.01		
		PASS	0.910	0.001	0.880	0.001	0.710	0.014		
Dataset	**#SN**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS Physic	1	GraphSage	0.850	0.001	0.780	0.004	0.590	0.003	0.947	0.901
		AS-GCN	0.240	0.125	0.260	0.139	0.140	0.021		
		FastGCN	0.950	0.001	0.940	0.001	0.840	0.002		
		PASS	0.920	0.003	0.850	0.004	0.650	0.004		
	3	GraphSage	0.940	0.002	0.900	0.001	0.720	0.006		
		AS-GCN	0.910	0.001	0.880	0.002	0.730	0.022		
		FastGCN	0.390	0.025	0.210	0.033	0.230	0.034		
		PASS	0.950	0.003	0.940	0.002	0.820	0.009		
	5	GraphSage	0.950	0.005	0.910	0.003	0.740	0.001		
		AS-GCN	0.930	0.001	0.900	0.001	0.760	0.001		
		FastGCN	0.090	0.036	0.150	0.048	0.260	0.033		
		PASS	0.960	0.002	0.940	0.003	0.830	0.020		

Table 10: **GNN performance on SCENARIO 4: distribution shift.**

Dataset	α	model	Original	std	Cluster	std	Generated	std	pearson	spearman
Cora	iid	GraphSage	0.830	0.010	0.820	0.003	0.760	0.024	0.867	0.832
		SGC	0.860	0.001	0.810	0.004	0.810	0.023		
		GAT	0.840	0.007	0.800	0.005	0.760	0.014		
		PPNP	0.840	0.007	0.800	0.008	0.810	0.016		
	0.01	GraphSage	0.790	0.007	0.780	0.010	0.650	0.011		
		SGC	0.820	0.003	0.780	0.002	0.710	0.001		
		GAT	0.780	0.007	0.760	0.005	0.680	0.005		
		PPNP	0.780	0.005	0.760	0.004	0.730	0.001		
	0.3	GraphSage	0.730	0.010	0.730	0.003	0.660	0.012		
		SGC	0.790	0.003	0.720	0.002	0.700	0.01		
		GAT	0.760	0.003	0.700	0.019	0.650	0.016		
		PPNP	0.770	0.008	0.730	0.006	0.680	0.017		
Dataset	α	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Citeseer	iid	GraphSage	0.690	0.005	0.640	0.003	0.570	0.021	0.812	0.799
		SGC	0.710	0.004	0.650	0.001	0.590	0.017		
		GAT	0.680	0.016	0.650	0.003	0.580	0.011		
		PPNP	0.690	0.002	0.630	0.002	0.610	0.007		
	0.01	GraphSage	0.590	0.009	0.550	0.012	0.510	0.018		
		SGC	0.640	0.002	0.580	0.003	0.560	0.014		
		GAT	0.610	0.005	0.550	0.003	0.510	0.022		
		PPNP	0.610	0.010	0.550	0.010	0.540	0.02		
	0.3	GraphSage	0.610	0.006	0.580	0.002	0.500	0.02		
		SGC	0.660	0.003	0.560	0.002	0.530	0.012		
		GAT	0.650	0.007	0.560	0.005	0.510	0.003		
		PPNP	0.630	0.001	0.550	0.005	0.550	0.012		
Dataset	α	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Pubmed	iid	GraphSage	0.840	0.002	0.810	0.002	0.720	0.009	0.830	0.794
		SGC	0.860	0.001	0.820	0.000	0.730	0.005		
		GAT	0.840	0.005	0.810	0.002	0.720	0.014		
		PPNP	0.820	0.002	0.800	0.002	0.730	0.004		
	0.01	GraphSage	0.810	0.007	0.750	0.008	0.660	0.01		
		SGC	0.800	0.002	0.760	0.004	0.680	0.007		
		GAT	0.790	0.005	0.760	0.005	0.660	0.021		
		PPNP	0.770	0.004	0.760	0.006	0.680	0.008		
	0.3	GraphSage	0.770	0.007	0.720	0.005	0.620	0.014		
		SGC	0.770	0.003	0.730	0.000	0.660	0.003		
		GAT	0.750	0.014	0.700	0.002	0.630	0.008		
		PPNP	0.740	0.009	0.730	0.004	0.660	0.001		
Dataset	α	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Computer	iid	GraphSage	0.850	0.009	0.800	0.012	0.790	0.008	0.906	0.860
		SGC	0.870	0.004	0.790	0.004	0.800	0.003		
		GAT	0.840	0.003	0.790	0.008	0.800	0.012		
		PPNP	0.840	0.003	0.800	0.005	0.810	0.003		
	0.01	GraphSage	0.790	0.013	0.740	0.010	0.750	0.003		
		SGC	0.800	0.003	0.750	0.006	0.740	0.003		
		GAT	0.770	0.028	0.750	0.005	0.750	0.006		
		PPNP	0.770	0.015	0.750	0.003	0.760	0.007		
	0.3	GraphSage	0.750	0.020	0.710	0.015	0.690	0.019		
		SGC	0.760	0.004	0.710	0.005	0.710	0.006		
		GAT	0.760	0.003	0.720	0.010	0.700	0.006		
		PPNP	0.740	0.004	0.710	0.009	0.710	0.021		
Dataset	α	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Photo	iid	GraphSage	0.890	0.001	0.890	0.002	0.910	0.003	0.771	0.847
		SGC	0.890	0.005	0.890	0.002	0.911	0.007		
		GAT	0.880	0.002	0.870	0.008	0.910	0.003		
		PPNP	0.880	0.002	0.900	0.002	0.910	0.006		
	0.01	GraphSage	0.880	0.014	0.850	0.016	0.850	0.012		
		SGC	0.880	0.008	0.860	0.006	0.840	0.015		
		GAT	0.860	0.011	0.850	0.002	0.830	0.007		
		PPNP	0.860	0.009	0.860	0.003	0.850	0.019		
	0.3	GraphSage	0.830	0.011	0.860	0.018	0.830	0.009		
		SGC	0.850	0.013	0.820	0.002	0.790	0.017		
		GAT	0.840	0.015	0.850	0.027	0.820	0.006		
		PPNP	0.860	0.015	0.860	0.007	0.850	0.02		
Dataset	α	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS CS	iid	GraphSage	0.870	0.004	0.880	0.001	0.850	0.011	0.792	0.751
		SGC	0.870	0.006	0.880	0.002	0.850	0.012		
		GAT	0.869	0.001	0.860	0.003	0.830	0.007		
		PPNP	0.870	0.006	0.880	0.002	0.840	0.008		
	0.01	GraphSage	0.800	0.003	0.820	0.012	0.790	0.006		
		SGC	0.880	0.002	0.860	0.002	0.830	0.003		
		GAT	0.850	0.004	0.840	0.006	0.800	0.01		
		PPNP	0.840	0.003	0.860	0.001	0.830	0.003		
	0.3	GraphSage	0.820	0.008	0.850	0.007	0.800	0.005		
		SGC	0.870	0.002	0.850	0.001	0.840	0.003		
		GAT	0.850	0.008	0.840	0.003	0.810	0.006		
		PPNP	0.840	0.001	0.850	0.003	0.830	0.005		
Dataset	α	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
MS Physic	iid	GraphSage	0.930	0.002	0.930	0.002	0.840	0.008	0.925	0.815
		SGC	0.920	0.001	0.920	0.001	0.840	0.007		
		GAT	0.930	0.002	0.920	0.002	0.820	0.011		
		PPNP	0.930	0.005	0.930	0.000	0.840	0.007		
	0.01	GraphSage	0.830	0.033	0.850	0.004	0.760	0.019		
		SGC	0.840	0.004	0.820	0.005	0.740	0.015		
		GAT	0.870	0.007	0.840	0.011	0.780	0.009		
		PPNP	0.840	0.007	0.830	0.006	0.740	0.009		
	0.3	GraphSage	0.840	0.012	0.840	0.009	0.680	0.023		
		SGC	0.810	0.009	0.820	0.003	0.700	0.009		
		GAT	0.850	0.011	0.840	0.002	0.720	0.019		
		PPNP	0.810	0.012	0.830	0.004	0.700	0.009		

Table 11: **GNN performance on link prediction.**

Dataset	predictor	model	Original	std	Cluster	std	Generated	std	pearson	spearman
Cora	Dot	GCN	0.720	0.010	0.770	0.009	0.680	0.012	0.781	0.741
		SGC	0.710	0.025	0.760	0.005	0.660	0.016		
		GIN	0.820	0.015	0.760	0.016	0.650	0.022		
		GAT	0.810	0.002	0.810	0.007	0.730	0.015		
	MLP	GCN	0.540	0.005	0.620	0.012	0.510	0.01		
		SGC	0.530	0.016	0.590	0.042	0.510	0.006		
		GIN	0.530	0.012	0.690	0.016	0.630	0.017		
		GAT	0.550	0.003	0.660	0.013	0.610	0.034		
Dataset	**predictor**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Citeseer	Dot	GCN	0.690	0.007	0.740	0.009	0.650	0.026	0.808	0.824
		SGC	0.700	0.003	0.730	0.013	0.670	0.022		
		GIN	0.830	0.008	0.720	0.003	0.650	0.01		
		GAT	0.750	0.005	0.780	0.012	0.680	0.021		
	MLP	GCN	0.580	0.005	0.650	0.012	0.590	0.01		
		SGC	0.580	0.008	0.640	0.025	0.590	0.023		
		GIN	0.570	0.011	0.720	0.012	0.610	0.024		
		GAT	0.610	0.005	0.680	0.001	0.620	0.009		
Dataset	**predictor**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Pubmed	Dot	GCN	0.800	0.018	0.810	0.005	0.670	0.019	0.725	0.420
		SGC	0.790	0.002	0.780	0.006	0.660	0.004		
		GIN	0.800	0.008	0.760	0.008	0.650	0.009		
		GAT	0.860	0.003	0.850	0.007	0.720	0.008		
	MLP	GCN	0.760	0.003	0.770	0.012	0.640	0.017		
		SGC	0.770	0.006	0.770	0.006	0.610	0.008		
		GIN	0.750	0.004	0.790	0.014	0.660	0.004		
		GAT	0.750	0.004	0.850	0.019	0.660	0.011		
Dataset	**predictor**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Computer	Dot	GCN	0.790	0.010	0.850	0.026	0.810	0.008	0.652	0.559
		SGC	0.760	0.005	0.770	0.030	0.730	0.025		
		GIN	0.800	0.013	0.880	0.004	0.830	0.005		
		GAT	0.750	0.057	0.840	0.014	0.560	0.08		
	MLP	GCN	0.810	0.005	0.890	0.005	0.830	0.012		
		SGC	0.800	0.000	0.850	0.020	0.730	0.021		
		GIN	0.800	0.003	0.890	0.010	0.810	0.01		
		GAT	0.860	0.005	0.910	0.005	0.800	0.005		
Dataset	**predictor**	**model**	**Original**	**std**	**Cluster**	**std**	**Generated**	**std**	**pearson**	**spearman**
Amazon Photo	Dot	GCN	0.890	0.011	0.920	0.005	0.860	0.016	0.887	0.443
		SGC	0.810	0.014	0.840	0.015	0.780	0.011		
		GIN	0.810	0.007	0.910	0.006	0.880	0.002		
		GAT	0.530	0.023	0.740	0.151	0.660	0.134		
	MLP	GCN	0.870	0.006	0.930	0.006	0.890	0.001		
		SGC	0.840	0.010	0.900	0.012	0.810	0.015		
		GIN	0.850	0.006	0.930	0.002	0.870	0.004		
		GAT	0.910	0.007	0.930	0.004	0.850	0.007		

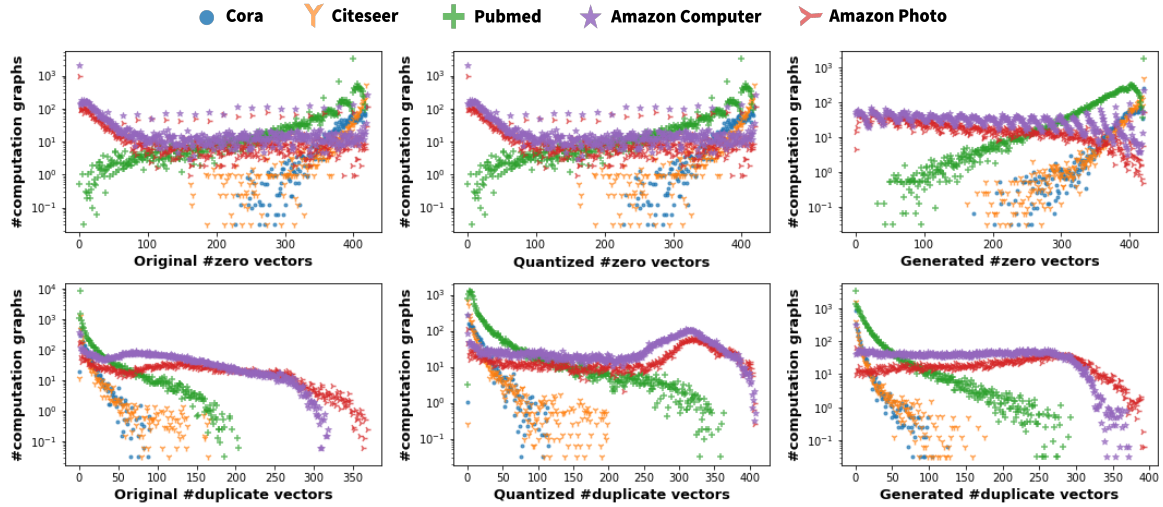


Figure 7: **CGT preserves distributions of graph statistics in generated graphs for each dataset:** While converting from original graphs to quantized graphs, CGT loses some of graph statistics information for k-anonymity privacy benefit. The variations given by CGT are presented as differences in distributions between quantized and generated graphs. X-axis denotes the number of zero vectors (z) and the number of duplicate vectors (d) per computation graph, respectively. Y-axis denotes the number of computation graphs with z zero vectors and d duplicate vectors, respectively.

where x_i denotes the d-dimensional feature vector of node v_i.

GCN (Kipf & Welling, 2016a). GCN models stack layers of first-order spectral filters followed by a nonlinear activation functions to learn node embeddings. When $h_i^{(l)}$ denotes the hidden embeddings of node v_i in the l-th layer, the simple and general form of GCNs is as follows:

$$h_i^{(l+1)} = \alpha(\frac{1}{n(i)} \sum_{j=1}^{n} a(v_i, v_j) h_j^{(l)} W^{(l)}), \quad l = 0, \ldots, L-1 \tag{1}$$

where $a(v_i, v_j)$ is set to 1 when there is an edge from v_i to v_j, otherwise 0. $n(i) = \sum_{j=1}^{n} a(v_i, v_j)$ is the degree of node v_i; $\alpha(\cdot)$ is a nonlinear function; $W^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l+1)}}$ is the learnable transformation matrix in the l-th layer with $d^{(l)}$ denoting the hidden dimension at the l-th layer. $h_i^{(0)}$ is set with the input node attribute x_i

GraphSage (Hamilton et al., 2017). GCNs require the full expansion of neighborhoods across layers, leading to high computation and memory costs. To circumvent this issue, GraphSage adds sampling operations to GCNs to regulate the size of neighborhood. We first recast Equation 1 as follows:

$$h_i^{(l+1)} = \alpha_{W^{(l)}}(\mathbb{E}_{j \sim p(j|i)}[h_j^{(l)}]), \quad l = 0, \ldots, L-1 \tag{2}$$

where we combine the transformation matrix $W^{(l)}$ into the activation function $\alpha_{W^{(l)}}(\cdot)$ for concision; $p(j|i) = \frac{a(v_i, v_j)}{n(i)}$ defines the probability of sampling v_j given v_i. Then we approximate the expectation by Monte-Carlo sampling as follows:

$$h_i^{(l+1)} = \alpha_{W^{(l)}}(\frac{1}{s} \sum_{j \sim p(j|i)}^{s} h_j^{(l)}), \quad l = 0, \ldots, L-1 \tag{3}$$

where s is the number of sampled neighbors for each node. Now, we can regulate the size of neighborhood using s, in other words, the computational footprint for each minibatch.

A.11.1. GNN MODELS USED IN THE BENCHMARK EFFECTIVENESS EXPERIMENT

We choose four different GNN models with different aggregation strategies to examine the effect of noisy edges on the aggregation strategies: GCN (Kipf & Welling, 2016a) with mean aggregator, GIN (Xu et al., 2018) with sum aggregator, SGC (Wu et al., 2019) with linear aggregator, and GAT (Veličković et al., 2017) with attention aggregator. We choose four different GNN models with different neighbor sampling strategies to examine the effect of noisy edges and number of sampled neighbor numbers on GNN performance: GraphSage (Hamilton et al., 2017) with random sampling, FastGCN (Chen et al., 2018) with heuristic layer-wise sampling, AS-GCN (Huang et al., 2018) with trainable layer-wise sampling, and PASS (Yoon et al., 2021) with trainable node-wise sampling. Finally, we choose four different GNN models to check their robustness to distribution shifts in training/test time, as the authors of the original paper (Zhu et al., 2021) chose for their baselines: GCN (Kipf & Welling, 2016a), SGC (Wu et al., 2019), GAT (Veličković et al., 2017), and PPNP (Klicpera et al., 2018).

We implement GCN, SGC, GIN, and GAT from scratch for the SCENARIO 1: noisy edges on aggregation strategies. For SCENARIOS 2 and 3: noisy edges and different sampling numbers on neighbor sampling, we use open

Table 12: **Ablation study**

Dataset	model	Original	Label	Position	Attention	All gone	Ours
Cora	**GCN**	0.860	0.510	0.710	0.580	0.570	0.760
	SGC	0.850	0.520	0.700	0.580	0.570	0.750
	GIN	0.850	0.510	0.620	0.600	0.570	0.750
	GAT	0.830	0.520	0.450	0.350	0.560	0.750
	GraphSage	0.750	0.210	0.590	0.320	0.600	0.500
	AS-GCN	0.120	0.170	0.240	0.070	0.140	0.110
	FastGCN	0.450	0.570	0.830	0.560	0.630	0.380
	PASS	0.800	0.470	0.750	0.410	0.600	0.540
	PPNP	0.840	0.555	0.850	0.743	0.584	0.810
Dataset	**model**	**Original**	**Label**	**Position**	**Attention**	**All gone**	**Ours**
Citeseer	**GCN**	0.730	0.450	0.670	0.530	0.520	0.590
	SGC	0.730	0.460	0.640	0.530	0.530	0.580
	GIN	0.710	0.450	0.520	0.530	0.510	0.570
	GAT	0.710	0.460	0.210	0.590	0.530	0.570
	GraphSage	0.680	0.280	0.580	0.370	0.550	0.440
	AS-GCN	0.110	0.200	0.280	0.220	0.160	0.100
	FastGCN	0.370	0.530	0.860	0.610	0.610	0.330
	PASS	0.700	0.480	0.550	0.450	0.550	0.460
	PPNP	0.690	0.540	0.760	0.393	0.547	0.610
Dataset	**model**	**Original**	**Label**	**Position**	**Attention**	**All gone**	**Ours**
Pubmed	GCN	0.860	0.680	0.970	0.670	0.740	0.780
	SGC	0.860	0.680	0.970	0.580	0.740	0.780
	GIN	0.830	0.670	0.990	0.670	0.740	0.770
	GAT	0.860	0.690	0.940	0.120	0.740	0.780
	GraphSage	0.780	0.460	0.360	0.920	0.740	0.600
	AS-GCN	0.250	0.320	0.200	0.770	0.360	0.260
	FastGCN	0.480	0.670	0.560	0.650	0.740	0.440
	PASS	0.860	0.690	0.330	1.000	0.740	0.660
	PPNP	0.820	0.687	0.190	0.997	0.736	0.730

source implementations of each GNN model, ASGCN [3], FastGCN [4], and PASS [5], uploaded by the original authors. Finally, for SCENARIO 4: distribution shift, we use GCN, SGC, GAT, and PPNP implemented by (Zhu et al., 2021) using DGL library [6].

A.12. Architecture of Computation Graph Transformer

Given a sequence $\mathbf{s} = [s_1, \cdots, s_T]$, the M-layered transformer maximizes the likelihood under the forward auto-regressive factorization as follow:

$$\max_{\theta} \log p_\theta(\mathbf{s}) = \sum_{t=1}^{T} \log p_\theta(s_t|\mathbf{s}_{<t}) = \sum_{t=1}^{T} \log \frac{exp(q_\theta^{(L)}(\mathbf{s}_{1:t-1})^\top e(s_t))}{\sum_{s' \neq s_t} exp(q_\theta^{(L)}(\mathbf{s}_{1:t-1})^\top e(s'))}$$

where node embedding $e(s_t)$ maps discrete input id s_t to a randomly initialized trainable vector, and query embedding $q_\theta^{(L)}(\mathbf{s}_{1:t-1})$ encodes information until $(t-1)$-th token in the sequence. Query embedding $q_t^{(l)}$ is computed with context embeddings $\mathbf{h}_{1:t-1}^{(l-1)}$ of previous $t-1$ tokens and query embedding $q_t^{(l-1)}$ from the previous layer. Context embedding $h_t^{(l)}$ is computed from $\mathbf{h}_{1:t}^{(l-1)}$, context embeddings of previous $t-1$ tokens and t-th token from the previous layer. Note that, while the query embeddings have access only to the previous context embeddings $\mathbf{h}_{1:t-1}^{(l)}$, the context embeddings attend to all tokens $\mathbf{h}_{1:t}^{(l)}$. The context embedding $h_t^{(0)}$ is initially encoded by node embeddings $e(s_t)$ and position embedding $p_{l(t)}$ that encodes the location of each token in the sequence. The query embedding is initialized with a trainable vector and label embeddings y_{s_1} as shown in Figure 3. This two streams (query and context) of self-attention layers are stacked M time and predict the next tokens auto-regressively.

A.13. Differentially Private k-means and SGD algorithms

Given a set of data points, k-means clustering identifies k points, called cluster centers, by minimize the sum of distances of the data points from their closest cluster center. However, releasing the set of cluster centers could potentially leak information about particular users. For instance, if a particular data point is significantly far from the rest of the points, so the k-means clustering algorithm returns this single point as a cluster center. Then sensitive information about this single point could be revealed. To address this, DP k-means clustering algorithm (Chang et al., 2021) is designed within the framework of differential privacy. To generate the private core-set, DP k-means partitions the points into buckets of similar points then replaces each bucket by a single weighted point, while adding noise to both the counts and averages of points within a bucket.

[3] https://github.com/huangwb/AS-GCN

[4] https://github.com/matenure/FastGCN

[5] https://github.com/linkedin/PASS-GNN

[6] https://github.com/GentleZhu/Shift-Robust-GNNs

Training a model is done through access to its parameter gradients, i.e., the gradients of the loss with respect to each parameter of the model. If this access preserves differential privacy of the training data, so does the resulting model, per the post-processing property of differential privacy. To achieve this goal, DP stochastic gradient descent (DP-SGD) (Song et al., 2013) modifies the minibatch stochastic optimization process to make it differentially private.

We use the open source implementation of DP k-means provided by Google's differential privacy libraries [7]. We extend implementations of DP SGD provided by a public differential library Opacus [8].

A.14. Privacy-enhanced graph synthesis

Various privacy-enhanced graph synthesis (Friedman & Schuster, 2010; Proserpio et al., 2012; Qin et al., 2017; Yang et al., 2020; Xiao et al., 2014; Sala et al., 2011) has been proposed to ensure differentially-private (DP) (Dwork, 2008) graph sharing. However, most of them are limited to small-scaled graphs using a few heuristic rules, while all of them do not consider node attributes and labels in their graph generation process (Xiao et al., 2014; Sala et al., 2011). Some GNN models have been proposed with DP guarantees (Olatunji et al., 2021; Sajadmanesh & Gatica-Perez, 2021), but this line of work concerns the *models* and not the *graphs*, and is therefore outside of our scope.

A.15. Experimental settings

All experiments were conducted on the same p3.2xlarge Amazon EC2 instance. We run each experiment three times and report the mean and standard deviation.

Dataset: We evaluate on seven public datasets — three citation networks (Cora, Citeseer, and Pubmed) (Sen et al., 2008), two co-purchase graphs (Amazon Computer and Amazon Photo) (Shchur et al., 2018), and two co-authorship graph (MS CS and MS Physic) (Shchur et al., 2018). We use all nodes when training CGT. For GNN training, we split $50\%/10\%/40\%$ of each dataset into the training/validation/test sets, respectively. We report their statistics in Table 13. AmazonC and AmazonP denote Amazon COmputer and Amazon Photo datasets, respectively.

Baselines: For the molecule graph generative models, GraphAF, GraphDF, and GraphEBM, we extend implementations in a public domain adaptation library DIG (Liu et al., 2021). We extend implementations of VGAE [9], GraphVAE [10] from codes uploaded by the authors of (Kipf & Welling, 2016b; You et al., 2018).

Table 13: **Dataset statistics.**

Dataset	Nodes	Edges	Features	Labels
Cora	2,485	5,069	1,433	7
Citeseer	2,110	3,668	3,703	6
Pubmed	19,717	44,324	500	3
AmazonC	13,381	245,778	767	10
AmazonP	7,487	119,043	745	8
MS CS	18,333	81,894	6,805	15
MS Physic	34,493	247,962	8,415	5

Model architecture: For our Computation Graph Transformer model, we use 3-layered transformers for Cora, Citeseer, Pubmed, and Amazon Computer, 4-layered transformers for Amazon Photo and MS CS, and 5-layered transformers for MS Physic, considering each graph size. For all experiments to examine the benchmark effectiveness of our model in Section 5.4, we sample $s = 5$ neighbors per node. For graph statistics shown in Section 5.3, we sample $s = 20$ neighbors per node.

[7] https://github.com/google/differential-privacy/tree/main/python/dp_accounting

[8] https://github.com/pytorch/opacus

[9] https://github.com/tkipf/gae

[10] https://github.com/JiaxuanYou/graph-generation