Kernel Ridge Regression-Based Graph Dataset Distillation

Zhe Xu
University of Illinois Urbana-Champaign
USA
zhexu3@illinois.edu

Yuzhong Chen
Visa Research
USA
yuzchen@visa.com

Menghai Pan
Visa Research
USA
menpan@visa.com

Huiyuan Chen
Visa Research
USA
hchen@visa.com

Mahashweta Das
Visa Research
USA
mahdas@visa.com

Hao Yang
Visa Research
USA
haoyang@visa.com

Hanghang Tong
University of Illinois Urbana-Champaign
Illinois, USA
htong@illinois.edu

ABSTRACT

The huge volume of emerging graph datasets has become a double-bladed sword for graph machine learning. On the one hand, it empowers a myriad of graph neural networks (GNNs) with strong empirical performance. On the other hand, training modern graph neural networks on huge graph data is computationally expensive. How to distill the given graph dataset while retaining most of the trained models' performance is a challenging problem. Existing efforts approach this problem by solving meta-learning-based bilevel optimization objectives. A major hurdle lies in that the exact solutions of these methods are computationally intensive and thus, most, if not all, of them are solved by approximate strategies which in turn hurt the distillation performance. In this paper, inspired by the recent advances in neural network kernel methods, we adopt a kernel ridge regression-based meta-learning objective which has a feasible *exact* solution. However, the computation of graph neural tangent kernel is very expensive, especially in the context of dataset distillation. In response, we design a graph kernel, named LiteGNTK, tailored for the dataset distillation problem which is closely related to the classic random walk graph kernel. An effective model named Kernel ridge regression-based graph Dataset Distillation (KIDD) and its variants are proposed. KIDD shows high efficiency in both the forward and backward propagation processes. At the same time, KIDD shows strong empirical performance over 7 real-world datasets compared with the state-of-the-art distillation methods. Thanks to the ability to find the exact solution of the distillation objective, the learned training graphs by KIDD can sometimes even outperform the original whole training set with as few as 1.65% training graphs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599398

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → **Data mining**; • **Theory of computation** → **Graph algorithms analysis**.

KEYWORDS

graph machine learning; graph dataset distillation

ACM Reference Format:
Zhe Xu, Yuzhong Chen, Menghai Pan, Huiyuan Chen, Mahashweta Das, Hao Yang, and Hanghang Tong. 2023. Kernel Ridge Regression-Based Graph Dataset Distillation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599398

1 INTRODUCTION

Graph datasets are indispensable parts of any graph machine learning and graph mining tasks ranging from fraud detection on financial systems [55], and fake account detection on social networks [20], to drug discovery in bioinformatics [24]. Graph neural networks (GNNs), as a family of powerful graph machine learning tools, are becoming critical modules in many graph machine learning systems due to their flexibility and strong expressiveness.

However, similar to other neural network methods, training GNNs with higher model expressiveness usually requires graph datasets with increasing volume [21, 37] and accordingly more intensive computation resource consumption. This limitation naturally leads to a question: *How can we find small, synthetic yet informative datasets to train GNNs with a competitive performance against GNNs trained on large graph datasets?*

The inquiry into the above question has incubated an emerging area named dataset distillation (DD) [5, 46] or dataset condensation (DC) [57]. The core idea of the existing DD or DC methods is to approach the problem under the umbrella of meta-learning and formulate it as a bilevel optimization problem [6]. Specifically, their lower-level problems have training objectives of fitting the synthetic datasets, while the upper-level optimization aims to find the proper synthetic datasets. A variety of settings have been explored for the upper-level problem. For example, Wang et al. [46] set the upper-level problem as the validation loss over the given large dataset; Zhao et al. [57] design a gradient matching loss as the

upper-level problem between the gradients on the original dataset and on the synthetic dataset.

The vast majority of the existing works on DD or DC are for tabular data and image data. DD/DC on graph datasets has not been well-studied due to the complex graph structure. To the best of our knowledge, the only graph-level (i.e., a graph is viewed as a data point) dataset distillation work is DosCond [26], which adopts the aforementioned gradient matching strategy [57]. To overcome the huge computation cost for solving the bilevel optimization problem, DosCond [26] proposes a fast but aggressive approximation of the gradient matching loss [57], which only matches the gradients of graph classifiers at initialization. Therefore, there exists unexplored space for performance improvement from the inexact solution of the distillation objective, which is one of our main foci.

Different from previous works, this paper aims to obtain the *exact* solution of the bilevel dataset distillation objective while maintaining computational tractability. To avoid the heavy computation of the bi-level optimization problem, we select kernel ridge regression (KRR) as the classifier whose prediction has a closed form. For implementing KRR on graph-level tasks, a graph kernel is required. Our work selects the recent graph neural tangent kernel (GNTK) [9] for the KRR graph classifier because GNTK describes the training dynamics of GNNs [9]. We name the proposed method Kernel ridge regression-based graph Dataset Distillation (KIDD).

However, due to the inevitable computational intensity of GNTK, naively connecting it with KRR results in low efficiency. To obtain a practically efficient dataset distillation method, we propose a series of novel enhanced designs. A simplified version of GNTK, LiteGNTK, is developed by removing non-linear activations at certain layers. This LiteGNTK is able to avoid heavy matrix multiplications during the iterative update of the synthetic graphs. By exploiting the close relationship between LiteGNTK and random walk graph kernel [28, 29], we further propose a fast low-rank KIDD variant (KIDD-LR) that boosts the efficiency further. To handle cases where discrete graph topology is required, another variant named KIDD-D is proposed by applying the Gumbel-Max reparameterization trick [23, 36] into our fully differentiable model KIDD.

In comprehensive experiments, our KRR-based model KIDD shows very strong empirical performance over 7 real-world datasets compared with the state-of-the-art methods. In addition, the efficiency study shows our proposed KIDD enjoys comparable efficiency to DosCond, an approximate solver of the distillation objective. In general, the contributions of this paper are summarized as follows,

- **Distillation method.** We propose the LiteGNTK ridge regression-based distillation objective function, which has a feasible exact solution. This distillation method is named KIDD.
- **Model enhancements.** We investigate a series of model enhancements specifically designed for the graph dataset distillation scenario to speed up the computation and expand the functionality. As a result, two practically efficient variants of KIDD (i.e., KIDD-LR and KIDD-D) are proposed.
- **Empirical evaluation.** We conduct extensive experiments on 7 real-world datasets in terms of effectiveness and efficiency. Results show that KIDD-LR and KIDD-D outperform the state-of-the-art methods significantly and, impressively, with as few as 1.65% of the number of training graphs, our methods can even outperform the original whole dataset under certain settings. Also, KIDD-LR and KIDD-D show sufficient efficiency, comparable to or even faster than the current SOTA graph dataset distillation method, DosCond.

2 PROBLEM DEFINITION

This section introduces the main notations used throughout this paper. After that, a brief introduction to graph neural networks (GNNs) and graph neural tangent kernel (GNTK) is presented. Then, a formal problem definition is provided.

2.1 Notations

We use bold letters for matrices (e.g., $\mathbf{A}$) and column vectors (e.g., $\mathbf{u}$). We use [] as the indices of matrices/vectors. E.g., $\mathbf{A}[i, j]$ represents the entry of matrix $\mathbf{A}$ at the i-th row and j-th column. Superscript $\top$ denotes the transpose of matrices/vectors. We use appropriate subscripts to denote the properties of nodes, graphs, and a set of graphs. For example, $\mathbf{h}_u$ is the representation of node u, $\mathcal{V}_\mathcal{G}$ is the node set of the graph $\mathcal{G}$, and $\mathbf{y}_\mathcal{T}$ is the labels of a graph set $\mathcal{T}$. All the synthetic/distilled graphs and their components are accented with tildes. For example, we notate a synthetic graph as $\tilde{\mathcal{G}} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\}$ where $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{X}}$ are its adjacency matrix and node feature matrix, respectively. The main notations are presented in Table 1.

2.2 Graph Neural Network

The primary operations of GNNs are as follows.

Aggregate. To update the node representation of node u, the representations of node u's neighbors are aggregated by the `aggregate` function. A typical `aggregate` function, the summation (with a re-scaling factor of the node u, c_u), is $\mathbf{h}_u \leftarrow c_u \sum_{v \in \mathcal{N}_u \cup \{u\}} \mathbf{h}_v$, where the common choice of $\mathcal{N}_u$ is u's 1-hop neighbors.

Update. The representation of a node u can be updated by the `update` function. A simple example is a fully-connected layer with element-wise non-linearity σ (e.g., ReLU [17]) as $\mathbf{h}_u \leftarrow \frac{1}{\sqrt{m}}\sigma(\mathbf{W}\mathbf{h}_u)$, where the m is the output dimension of the $\mathbf{W}\mathbf{h}_u$.

Readout. For the graph-level tasks, the graph representation is aggregated over all the nodes in a graph $\mathcal{G}$ by the `readout` function. A typical `readout` function, the summation, is $\mathbf{h}_\mathcal{G} = \sum_{v \in \mathcal{V}_\mathcal{G}} \mathbf{h}_v$.

GNN variants on graph-level tasks are usually composed of multiple `aggregate` and `update` operations and end with an `readout` operation.

2.3 Graph Neural Tangent Kernel

Graph neural tangent kernel (GNTK) [9] is a graph kernel that describes infinitely wide multi-layer GNNs trained by gradient descent through the squared loss. Concretely, the tangent kernel of a GNN f is presented as

$$K_{\boldsymbol{\theta}}(\mathcal{G}, \mathcal{G}') = \left\langle \frac{\partial f(\boldsymbol{\theta}, \mathcal{G})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}, \mathcal{G}')}{\partial \boldsymbol{\theta}} \right\rangle, \tag{1}$$

where $\boldsymbol{\theta}$ denotes the set of trainable parameters of the GNN. If the width of the GNN is infinite (i.e., $m \to \infty$) and every trainable parameter is an i.i.d. Gaussian random variable, the expectation of the above tangent kernel can be explicitly computed as $K_{\mathsf{GNTK}}(\mathcal{G}, \mathcal{G}')$ [9] and it is named GNTK. Du. et al [9] provide a

Table 1: Symbols and Notations.

Symbol	Definition
$\mathcal{G}$	graph
$\mathcal{V}_{\mathcal{G}}$	node set of the graph $\mathcal{G}$
$\mathbf{A}$	adjacency matrix
$\mathbf{X}$	node feature matrix
y	graph label
$\mathcal{T}$	target graph training set
$\mathcal{S}$	synthetic graph training set
$n_{\mathcal{T}}$	number of target graphs
$n_{\mathcal{S}}$	number of synthetic graphs
$\mathbf{y}_{\mathcal{T}}$	label vector of the graph set $\mathcal{T}$
$\mathbf{y}_{\mathcal{S}}$	label vector of the graph set $\mathcal{S}$

recipe to translate the computation of GNN on node representations from a graph $\mathcal{G}$ into the computation of GNTK on a node covariance matrix from a pair of graphs $\mathcal{G}$ and $\mathcal{G}'$.

Given the graph pair $\mathcal{G}$ and $\mathcal{G}'$ with their node sets $\mathcal{V}_{\mathcal{G}}$ and $\mathcal{V}_{\mathcal{G}'}$, for a pair of nodes $u \in \mathcal{V}_{\mathcal{G}}$ and $u' \in \mathcal{V}_{\mathcal{G}'}$, if we slightly abuse the u and u' as the indices of the matrix, the initial covariance matrix $\Sigma_{\mathcal{G},\mathcal{G}'}$ and the initial GNTK matrix $\Theta_{\mathcal{G},\mathcal{G}'}$ can be computed as

$$\Sigma_{\mathcal{G},\mathcal{G}'}[u,u'] = \Theta_{\mathcal{G},\mathcal{G}'}[u,u'] = \mathbf{x}_u^\top \mathbf{x}_{u'}, \tag{2}$$

where $\mathbf{x}$ is the raw node feature. Then, the `aggregate` operation is translated by the GNTK recipe as

$$\Sigma_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow c_u c_{u'} \sum_{v\in\mathcal{N}_u\cup\{u\}} \sum_{v'\in\mathcal{N}_{u'}\cup\{u'\}} \Sigma_{\mathcal{G},\mathcal{G}'}[v,v'] \tag{3a}$$

$$\Theta_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow c_u c_{u'} \sum_{v\in\mathcal{N}_u\cup\{u\}} \sum_{v'\in\mathcal{N}_{u'}\cup\{u'\}} \Theta_{\mathcal{G},\mathcal{G}'}[v,v'] \tag{3b}$$

The `update` operation is translated in the following steps

$$\Lambda_{\mathcal{G},\mathcal{G}'}[u,u'] = \begin{pmatrix} \Sigma_{\mathcal{G},\mathcal{G}}[u,u] & \Sigma_{\mathcal{G},\mathcal{G}'}[u,u'] \\ \Sigma_{\mathcal{G}',\mathcal{G}}[u',u] & \Sigma_{\mathcal{G}',\mathcal{G}'}[u',u'] \end{pmatrix}, \tag{4a}$$

$$\dot{\Sigma}_{\mathcal{G},\mathcal{G}'}[u,u'] = \mathbb{E}_{(a,b)\sim\mathcal{N}(0,\Lambda_{\mathcal{G},\mathcal{G}'}[u,u'])} \dot{\sigma}(a)\dot{\sigma}(b), \tag{4b}$$

$$\Sigma_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow \mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\Lambda_{\mathcal{G},\mathcal{G}'}[u,u'])} \sigma(a)\sigma(b), \tag{4c}$$

$$\Theta_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow \Theta_{\mathcal{G},\mathcal{G}'}[u,u']\dot{\Sigma}_{\mathcal{G},\mathcal{G}'}[u,u'] + \Sigma_{\mathcal{G},\mathcal{G}'}[u,u'], \tag{4d}$$

where $\dot{\sigma}(\cdot)$ is the derivative of ReLU. Finally, the `readout` operation is translated as follows, which computes the final GNTK value between the graph pair $\mathcal{G}$ and $\mathcal{G}'$

$$K_{\mathsf{GNTK}}(\mathcal{G},\mathcal{G}') = \sum_{u\in\mathcal{V}_{\mathcal{G}},u'\in\mathcal{V}_{\mathcal{G}'}} \Theta_{\mathcal{G},\mathcal{G}'}[u,u']. \tag{5}$$

To be self-contained, the exact computation of Eq. (4b) and Eq. (4c) proposed by [3, 9] is provided in Appendix.

2.4 Graph Dataset Distillation

In this paper, we study the dataset distillation problem for graph classification tasks. Specifically, we aim to synthesize a small number of informative graphs to empower the training of graph classifiers. The problem is formally defined as follows.

Problem 1. *Graph dataset distillation*
***Given**: a set of target training graphs* $\mathcal{T} = \{(\mathcal{G}_i, y_i)\}_0^{n_{\mathcal{T}}-1}$.
***Find**: a set of synthetic training graphs* $\mathcal{S} = \{(\tilde{\mathcal{G}}_i, \tilde{y}_i)\}_0^{n_{\mathcal{S}}-1}$ *such that the GNN trained over* $\mathcal{S}$ *can obtain competitive performance with GNN trained over* $\mathcal{T}$.

The overall procedure is two-step. First, a distillation method (including a **distillation classifier**, introduced in Section 3) is applied to synthesize $\mathcal{S}$ from $\mathcal{T}$. Then, a **downstream classifier** is trained over $\mathcal{S}$ and reports its performance on a test set $\mathcal{U}$ which has no overlap with $\mathcal{S}$ or $\mathcal{T}$. The downstream classifier's test performance is the metric of the distilled training set $\mathcal{S}$, and in this paper, the downstream classifier is selected from graph-level GNNs (e.g., GIN [50]) and will be introduced in detail in Section 4.

3 PROPOSED METHOD

This section introduces the objective formulation, the proposed model (KIDD), and the corresponding enhancements.

3.1 Optimization Objective

An ideal distilled graph dataset $\mathcal{S}$ for the target graph dataset $\mathcal{T}$ should minimize the following optimization objective.

$$\min_{\mathcal{S}} \quad |\mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_{\mathcal{S}})) - \mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_{\mathcal{T}}))|, \tag{6a}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*_{\mathcal{S}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{S}, f(\boldsymbol{\theta})), \tag{6b}$$

$$\boldsymbol{\theta}^*_{\mathcal{T}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta})), \tag{6c}$$

where $\mathcal{U}$ is the test set sampled from the true graph distribution. The above formula suggests the graph classifiers trained on the target dataset (i.e., $f(\boldsymbol{\theta}^*_{\mathcal{T}})$ from Eq.(6c)) and on the synthetic dataset (i.e., $f(\boldsymbol{\theta}^*_{\mathcal{S}})$ from Eq.(6b)) should have similar expected test loss (i.e., Eq. (6a)). Notice here the graph classifier f is for the distillation purpose, i.e., synthesizing the dataset $\mathcal{S}$. Thus, we name f as the **distillation classifier** in this paper. Naturally, Eq. (6a) implies *the best choice of the distillation classifier f should be the same as the downstream classifier.* That is because, if the downstream classifier is g, the test error $\mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, g(\boldsymbol{\theta}^*_{\mathcal{S}}))$ is minimized when $f = g$, considering (1) $|\mathcal{T}| \gg |\mathcal{S}|$ and (2) empirically, $\mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_{\mathcal{S}})) \geq \mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_{\mathcal{T}}))$.

However, as the true graph distribution is not accessible, a feasible optimization goal is to replace the unknown test set $\mathcal{U}$ with the large target training set $\mathcal{T}$ as follows.

$$\min_{\mathcal{S}} \quad |\mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{S}})) - \mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{T}}))|, \tag{7}$$

where $\boldsymbol{\theta}^*_{\mathcal{T}}$ and $\boldsymbol{\theta}^*_{\mathcal{S}}$ are from Eq. (6c) and Eq. (6b). As the $\boldsymbol{\theta}^*_{\mathcal{T}}$ is the minimizer over the $\mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}))$, to minimize Eq. (7), our objective is equivalent to minimizing $\mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{S}}))$. The objective function can be re-written as follows,

$$\min_{\mathcal{S}} \quad \mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{S}})), \tag{8a}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*_{\mathcal{S}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{S}, f(\boldsymbol{\theta})), \tag{8b}$$

which can be interpreted as finding a synthetic training set $\mathcal{S}$ such that the trained model $f(\boldsymbol{\theta}^*_{\mathcal{S}})$ over $\mathcal{S}$ has a small loss over the validation set $\mathcal{T}$. The above objective is a bilevel optimization problem [6] whose exact solutions [12, 13, 34, 35, 41, 44] is computationally expensive or even intractable, especially when the objective functions are not convex. Thus, even though the best distillation classifier f is

the downstream GNN classifier itself, its non-convex optimization objective makes the distillation problem costly to be solved. In this paper, we resort to a kernel-based method that renders a tractable and exact solution to the above objective function.

3.2 Graph Kernel Ridge Regression

To avoid the expensive hyper-gradient computation of the bilevel optimization problem, a strategy is that if the lower-level problem has a closed-form solution, plugging the lower-level solution into the upper-level objective can largely simplify the optimization objective [33, 39] into a single-level problem. Kernel ridge regression (KRR) can yield such a closed-form solution. If f is instantiated as the KRR and the squared loss is applied, the Eq. (8a) and Eq. (8b) can be instantiated as

$$\min_{\mathcal{S}} \quad \mathcal{L}_{\text{KRR}} = \frac{1}{2}||\mathbf{y}_{\mathcal{T}} - \mathbf{K}_{\mathcal{TS}}(\mathbf{K}_{\mathcal{SS}} + \epsilon\mathbf{I})^{-1}\mathbf{y}_{\mathcal{S}}||^2, \tag{9}$$

where $\epsilon > 0$ is a KRR hyper-parameter, $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ are the kernel matrices. E.g., $\mathbf{K}_{\mathcal{TS}}[i,j] = K(\mathcal{G}_i, \tilde{\mathcal{G}}_j)$, $\mathbf{K}_{\mathcal{SS}}[i,j] = K(\tilde{\mathcal{G}}_i, \tilde{\mathcal{G}}_j)$, $\mathcal{G}_i \in \mathcal{T}$, $\tilde{\mathcal{G}}_i$, $\tilde{\mathcal{G}}_j \in \mathcal{S}$ with K as a specific kernel function (e.g., random walk graph kernel [28, 29]). $\mathbf{y}_{\mathcal{T}}$ and $\mathbf{y}_{\mathcal{S}}$ are the concatenated graph labels from $\mathcal{T}$ and $\mathcal{S}$.

However, there are two concerns about the above optimization objective. First (*model gap*), in terms of expressive power and empirical performance, GNNs outperform most, if not all, classic graph kernel methods on graph classification tasks [50]. It leads to a concern that *whether the graph dataset $\mathcal{S}$ distilled through the graph kernel-based kernel ridge regression can empower the training of downstream GNNs*. Second (*informative distillation*), as the size of the distilled graph dataset is small, how to ensure the distilled graphs are sufficiently informative to capture the critical information from the original training set.

As a response to the first concern and to bridge the gap between the distillation classifier and the downstream GNN classifier, a recent graph kernel is adopted which is named graph neural tangent kernel (GNTK) [9]. GNTK (1) measures the graph pair similarity by mapping a graph into an infinite-dimensional GNN gradient vector and (2) describes the training dynamics of the corresponding GNN. Thus, GNTK-based kernel ridge regression is promising to distill generalizable graph datasets for downstream GNN classifiers. The specific computation of GNTK is introduced in Section 2.

As a response to the second concern and to ensure informative distilled graphs, we propose a distillation regularization term as

$$\mathcal{L}_{\texttt{reg}} = ||\hat{\mathbf{K}}_{\mathcal{SS}} - \mathbf{I}||_F^2. \tag{10}$$

The final objective $\mathcal{L}$ is the weighted sum of $\mathcal{L}_{\text{KRR}}$ and $\mathcal{L}_{\texttt{reg}}$ as

$$\min_{\mathcal{S}} \quad \mathcal{L} = \mathcal{L}_{\text{KRR}} + \gamma\mathcal{L}_{\texttt{reg}}. \tag{11}$$

$\hat{\mathbf{K}}_{\mathcal{SS}}$ is the normalized kernel matrix between synthetic graphs whose entry $\hat{\mathbf{K}}_{\mathcal{SS}}[i,j] = \frac{\mathbf{K}_{\mathcal{SS}}[i,j]}{\sqrt{\mathbf{K}_{\mathcal{SS}}[i,i]}\sqrt{\mathbf{K}_{\mathcal{SS}}[j,j]}}$. γ is a hyper-parameter to trade-off the KRR classification loss $\mathcal{L}_{\text{KRR}}$ and the regularization loss $\mathcal{L}_{\texttt{reg}}$. The key idea of $\mathcal{L}_{\texttt{reg}}$ is to force the normalized kernel matrix $\hat{\mathbf{K}}_{\mathcal{SS}}$ to be an identity matrix so that the synthetic graphs are more orthogonal to each other in the kernel space.

Given the optimization objective as Eq. (11) shows, the gradient of the objective with respect to the graphs from $\mathcal{S}$ are computed, i.e., $\frac{\partial\mathcal{L}}{\partial\tilde{\mathcal{G}}_i}$, $\forall\tilde{\mathcal{G}}_i \in \mathcal{S}$. We provide some details here. First, in our implementation, we compute the gradient with respect to the graph adjacency matrix $\tilde{\mathbf{A}}$ and node feature matrix $\tilde{\mathbf{X}}$ for all the graphs from $\mathcal{S} = \{(\tilde{\mathcal{G}}_i, \tilde{y}_i)\}_0^{n_{\mathcal{S}}-1}$. Second, any gradient-based optimizer can be applied, e.g., Adam [30]. Third, the synthetic set $\mathcal{S}$ can be initialized by sampling the target set $\mathcal{T}$ or fully initialized randomly. In this work, we initialize the synthetic set $\mathcal{S}$ by sampling $\mathcal{T}$. We name our proposed graph dataset distillation method KIDD. Next, we present the enhancements designed for KIDD to improve its efficiency greatly and handle discrete graph structures.

3.3 Model Enhancements

The existing computation of GNTK [9] is still expensive, especially for our gradient descent-based dataset distillation scenario. That is because, in every iteration, the kernel matrices $\mathbf{K}_{\mathcal{SS}}$ and $\mathbf{K}_{\mathcal{TS}}$ need to be re-computed. Additionally, it is non-trivial to synthesize discrete graph topology by the gradient descent-based method, which is required in certain cases. This section introduces several enhancements that systematically improve KIDD's computational feasibility and functionality.

LiteGNTK. As we introduced in Section 2, Du. et al [9] provides a recipe to translate a specific instantiation of GNN into its corresponding GNTK. The GNTK used in the existing literature [9] is based on the typical GNNs with a non-linear activation function at every layer (e.g., GCN [31] and GIN [50]). Hence, every layer of their corresponding GNTK instantiations contains a translated `update` operation (i.e., Eq. (4a)-(4d)). Recently, extensive empirical studies on GNNs show that non-linearity is not a necessity for every GNN layer, and removing the non-linearity from the last several layers can speed up the computation [48] without sacrificing or even improving the model performance [18]. Thus, we propose to remove the `update` operation from all the GNTK layers except the first one and use the following LiteGNTK for our kernel ridge regression-based graph dataset distillation. Given two graphs $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and $\tilde{\mathcal{G}} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\}$, the LiteGNTK $K_{\texttt{LiteGNTK}}$ between this graph pair is computed as

$$\Sigma_{\mathcal{G},\tilde{\mathcal{G}}} = \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} = \mathbf{X}\tilde{\mathbf{X}}^\top, \tag{12a}$$

$$\Sigma_{\mathcal{G},\tilde{\mathcal{G}}}, \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \leftarrow \texttt{GNTK-update}(\Sigma_{\mathcal{G},\tilde{\mathcal{G}}}, \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}), \tag{12b}$$

$$\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \leftarrow (\mathbf{c}_{\mathcal{G}}\mathbf{c}_{\tilde{\mathcal{G}}}^\top) \odot \left(\mathbf{A}^k\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{A}}^\top)^k\right), \tag{12c}$$

$$K_{\texttt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}) = \sum_{u,\tilde{u}} \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}[u,\tilde{u}], \tag{12d}$$

where $\odot$ is the Hadamard product; `GNTK-update` is an alias of Eq. (4a)-(4d); $\mathbf{c}_{\mathcal{G}} = \texttt{vec}(\{c_u^k | u \in \mathcal{G}\})$ is the scaling vector whose elements are the k-powered re-scaling factor of every node from $\mathcal{G}$; $\mathbf{c}_{\tilde{\mathcal{G}}}$ is constructed similarly by the nodes from $\tilde{\mathcal{G}}$. Finally, the kernel value between the graph $\mathcal{G}$ and $\tilde{\mathcal{G}}$ are computed by the Eq. (12d) which is the same as the GNTK `readout` function. LiteGNTK is used for all our proposed distillation models.

Advantages of LiteGNTK. First, the original GNTK [9] contains non-linearity in each of the k layers, which requires the computation of $\mathbf{A}\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\tilde{\mathbf{A}}^\top$ for k times during every update of the synthetic graphs. In comparison, for LiteGNTK, $\mathbf{A}^k$ and $\tilde{\mathbf{A}}^k$ can be precomputed, especially for the $\mathbf{A}$s from the target dataset $\mathcal{T}$ which are

not updated throughout the whole distillation procedure. Second, the LiteGNTK is closely related to the random walk graph kernel [28, 29] which is shown in our following proposition. As both LiteGNTK and random walk graph kernel include power iterations of adjacency matrices, it reveals the possibility to develop a faster LiteGNTK in light of the speedup of the random walk graph kernel.

Proposition 1. *LiteGNTK is a generalized instantiation of the random walk graph kernel [28, 29].*

Proof. Random walk graph kernel (RWK) counts the number of length-k common paths between graph $\mathcal{G}$ and $\tilde{\mathcal{G}}$ as

$$K_{\mathtt{RWK}}(\mathcal{G},\tilde{\mathcal{G}}) = (\mathbf{q}\otimes\tilde{\mathbf{q}})^\top(\mathbf{A}\otimes\tilde{\mathbf{A}})^k(\mathbf{p}\otimes\tilde{\mathbf{p}}), \tag{13}$$

where $\mathbf{p}$ ($\tilde{\mathbf{p}}$) and $\mathbf{q}$ ($\tilde{\mathbf{q}}$) are the starting and stopping probability vectors of graph $\mathcal{G}$ ($\tilde{\mathcal{G}}$). Thus, Eq. (13) can be written as

$$K_{\mathtt{RWK}} \underset{(1)}{=} (\mathbf{q}\otimes\tilde{\mathbf{q}})^\top(\mathbf{A}\otimes\tilde{\mathbf{A}})^{(k-1)}\mathtt{vec}\Big(\mathbf{A}\Big(\mathtt{vec}^{-1}(\mathbf{p}\otimes\tilde{\mathbf{p}})\Big)\tilde{\mathbf{A}}^\top\Big), \tag{14a}$$

$$\underset{(2)}{=} (\mathbf{q}\otimes\tilde{\mathbf{q}})^\top\mathtt{vec}\Big(\mathbf{A}^k(\mathbf{p}\tilde{\mathbf{p}}^\top)(\tilde{\mathbf{A}}^\top)^k\Big), \tag{14b}$$

$$\underset{(3)}{=} \mathbf{q}^\top\mathbf{A}^k(\mathbf{p}\tilde{\mathbf{p}}^\top)(\tilde{\mathbf{A}}^\top)^k\tilde{\mathbf{q}}, \tag{14c}$$

$$\underset{(4)}{=} \sum_{u,\tilde{u}}\Big((\mathbf{q}\tilde{\mathbf{q}}^\top)\odot\Big(\mathbf{A}^k(\mathbf{p}\tilde{\mathbf{p}}^\top)(\tilde{\mathbf{A}}^\top)^k\Big)\Big)[u,\tilde{u}], \tag{14d}$$

where $\otimes$ is the Kronecker product, $\odot$ is the Hadamard product, vec is the vectorization operator, and $\mathtt{vec}^{-1}$ is the inverse operator of vec (i.e., reshaping a vector into a matrix). Note here the vec and $\mathtt{vec}^{-1}$ follow the row-first order as many scientific computing packages do (e.g., PyTorch's reshape function). The above step (1), (2), and (3) are based on the property of the Kronecker product: $(\mathbf{A}\otimes\mathbf{B})\mathbf{v} = \mathtt{vec}\Big(\mathbf{A}\Big(\mathtt{vec}^{-1}(\mathbf{v})\Big)\mathbf{B}^\top\Big)$. Step (4) is because $\mathbf{v}^\top\mathbf{A}\mathbf{v} = \sum_{i,j}\mathbf{v}[i]\mathbf{A}[i,j]\mathbf{v}[j] = \sum_{i,j}\mathbf{v}[i]\mathbf{v}[j]\mathbf{A}[i,j] = \sum_{i,j}\Big((\mathbf{v}\mathbf{v}^\top)\odot\mathbf{A}\Big)[i,j]$. Here, the matrix $\mathbf{p}\tilde{\mathbf{p}}^\top$ describes the node pair co-starting probability from graphs $\mathcal{G}$ and $\tilde{\mathcal{G}}$ which can be generalized as the GNTK-updated node pair co-variance matrix $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}$ from Eq. (12b). The matrix $\mathbf{q}\tilde{\mathbf{q}}^\top$ is a node pair co-stopping probability matrix which can be generalized as the node pair weighting matrix $\mathbf{c}_{\mathcal{G}}\mathbf{c}_{\tilde{\mathcal{G}}}^\top$ from Eq. (12c). Thus, we conclude that the LiteGNTK $K_{\mathtt{LiteGNTK}}$ is the generalization of the random walk graph kernel $K_{\mathtt{RWK}}$. □

Low-Rank Speed-Up. Proposition 1 reveals the connection between the random walk graph kernel and the LiteGNTK, which inspires us to apply the fast strategy [28] proposed for the random walk graph kernel into the computation of LiteGNTK. Real-world graphs are known for their intrinsic low-rank topology [10], which can significantly speed up the power iterations of the adjacency matrices using their low-rank representations. To be specific, for learning the topology of synthetic graphs, instead of optimizing the objective Eq. (11) with respect to the synthetic graph's adjacency matrix $\tilde{\mathbf{A}}$, we directly optimize its low-rank decomposed matrix $\tilde{\mathbf{U}},\ \tilde{\mathbf{V}}\in\mathbb{R}^{n\times r}$ where r is the rank of the synthetic graphs.

Notice that in this paper, we retain the adjacency matrices of the original training graphs $\mathcal{G}\in\mathcal{T}$ and only learn the low-rank matrices of the synthetic graphs $\tilde{\mathcal{G}}\in\mathcal{S}$. There are 2 reasons: (1) the original training graphs are the distillation target, whose topology information should be kept intact; (2) the original training graphs are usually sparse but the synthetic graphs could be dense due to the gradient descent update, which leads to heavy computation (e.g., matrix multiplication). Since the $\mathbf{K}_{\mathcal{TS}}$ has much more entries than the matrix $\mathbf{K}_{\mathcal{SS}}$, in the following part, we analyze the computation of the kernel value between $\mathcal{G}=\{\mathbf{A},\mathbf{X}\}\in\mathcal{T}$ and $\tilde{\mathcal{G}}=\{\tilde{\mathbf{U}},\tilde{\mathbf{V}},\tilde{\mathbf{X}}\}\in\mathcal{S}$, i.e., an entry of the matrix $\mathbf{K}_{\mathcal{TS}}$. For computing entries of $\mathbf{K}_{\mathcal{SS}}$, it can be analyzed similarly and is omitted for brevity.

Given $\mathcal{G}=\{\mathbf{A},\mathbf{X}\}\in\mathcal{T}$ and $\tilde{\mathcal{G}}=\{\tilde{\mathbf{U}},\tilde{\mathbf{V}},\tilde{\mathbf{X}}\}\in\mathcal{S}$, to compute their corresponding LiteGNTK value, Eq. (12c) is modified as follows,

$$\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \leftarrow (\mathbf{c}_{\mathcal{G}}\mathbf{c}_{\tilde{\mathcal{G}}}^\top)\odot\Big(\mathbf{A}^k\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k\Big), \tag{15}$$

while Eq. (12a), (12b), (12d) stay unchanged. The following lemma shows such a minor change can greatly improve the model's efficiency during both the forward computation and the gradient backward propagation. Notice that the following analysis focuses on the computation efficiency of the key operations containing the synthetic graph topology variables $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$.

Lemma 1. *(Time Complexity) Assume both $\mathcal{G}$ and $\tilde{\mathcal{G}}$ have n nodes, i.e., $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\in\mathbb{R}^{n\times n}$, the time complexity of computing $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k$ is $O(rn^2)$. The time complexity of computing $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{U}}}$ and $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{V}}}$ is $O(r^3n^3)$.*

Proof. See appendix. □

As a comparison, without this low-rank speed-up technique, the time complexity of computing $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\tilde{\mathbf{A}}^k$ is $O(kn^3)$ and time complexity of computing $\frac{\partial\tilde{\mathbf{A}}^k}{\partial\tilde{\mathbf{A}}}$ is $O(kn^4)$ if $\tilde{\mathbf{A}}$ is a dense matrix after gradient descent-based updating. Thus, considering $r\ll n$, our proposed low-rank variant can significantly speed up the computation. In addition, clearly, the space complexity for storing every synthetic graph's topology drops to $O(2nr)$ if the low-rank technique is applied; otherwise, it is $O(n^2)$.

Benefiting from the close connection between the random walk graph kernel and LiteGNTK, the following lemma gives an error bound on applying the low-rank technique to LiteGNTK. For brevity, the lemma 2 only analyzes the case where synthetic graphs are undirected (i.e., $\tilde{\mathbf{A}}$ is symmetric).

Lemma 2. *(Error Bound) Given a target graph $\mathcal{G}=\{\mathbf{A},\mathbf{X}\}$, a synthetic graph $\tilde{\mathcal{G}}=\{\tilde{\mathbf{A}},\tilde{\mathbf{X}}\}$ and $\tilde{\mathcal{G}}$'s rank-r representation $\tilde{\mathcal{G}}_r=\{\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top,\tilde{\mathbf{X}}\}$, if we assume both $\mathcal{G}$ and $\tilde{\mathcal{G}}$ have n nodes, $\mathcal{G}$ has m edges, and $\mathbf{c}_{\mathcal{G}}=\mathbf{c}_{\tilde{\mathcal{G}}}=\mathbf{c}_{\tilde{\mathcal{G}}_r}=\mathbf{1}$ are all-one vectors, the error of the LiteGNTK value after applying the low-rank speed-up can be bounded by*

$$\mathbf{K}_{\mathtt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}) - \mathbf{K}_{\mathtt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}_r) \le nm^{\frac{k}{2}}||\mathbf{\Theta}||_F\sum_{i=r+1}^{n}|\tilde{\lambda}_i^{\,k}|, \tag{16}$$

where $\mathbf{\Theta}=\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}=\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}_r}$ is the output of Eq. (12b) and $\tilde{\lambda}_i$ is the i-th largest eigenvalue of $\tilde{\mathbf{A}}$.

Proof. See appendix. □

Our distillation model equipped with this low-rank speed-up technique is named KIDD-LR, and a step-by-step algorithm to distill graph dataset is presented in Appendix - Algorithm 1. In addition, as Lemma 2 shows if the low-rank assumption of the synthetic

graphs holds, the proposed KIDD-LR still provides an exact solution; otherwise, it provides an approximate solution.

Discrete Synthetic Graphs. In spite of the great efficacy of our proposed low-rank speed-up method, the synthetic graph topology $\tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top$ is weighted and even needs clipping to be non-negative in the inference phase. For the case where discrete unweighted graphs are needed, i.e., all the entries all either 0 or 1, a discrete variant of our method is proposed. The strategy is to model every pair of nodes as an independent Bernoulli variable [14, 26, 45] (e.g., $b_{u,v} \in [0, 1]$) such that $\tilde{\mathbf{A}}[u, v] \sim \texttt{Bernoulli}(b_{u,v})$. As the sampling process is not differentiable, we utilize the Gumbel-Max reparametrization trick [23, 26, 36], and the adjacency matrix is computed as

$$\tilde{\mathbf{A}}[u, v] = \texttt{sigmoid}((\log \delta - \log(1 - \delta) + b_{u,v})/\tau), \tag{17}$$

where $\delta \sim \texttt{Uniform}(0, 1)$, and τ is a temperature hyperparameter. If $\tau \to 0$, $\tilde{\mathbf{A}}[u, v]$ will be binary. Then, instead of modeling graph topology by its adjacency matrix, we can describe it by a corresponding Bernoulli parameter matrix $\tilde{\mathbf{B}}$ whose entries are edge-existing Bernoulli variables, i.e., $\tilde{\mathbf{B}}[u, v] = b_{u,v}$. As the gradient $\frac{\partial \tilde{\mathbf{A}}[u,v]}{\partial \tilde{\mathbf{B}}[u,v]}$ is well-defined, we can optimize $\tilde{\mathbf{B}}$ in an end-to-end fashion by the gradient descent as we introduced at the end of Section 3.2. During the inference phase, we can set $\tilde{\mathbf{A}}[u, v] = 1$ if $\texttt{sigmoid}(b_{u,v}) > 0.5$; otherwise, $\tilde{\mathbf{A}}[u, v] = 0$.

Our distillation model equipped with this Gumbel-Max technique is named KIDD-D and a detailed algorithm to use KIDD-D distill graph dataset is presented in Appendix - Algorithm 2. Notice that even though we can apply a similar low-rank decomposition trick to decompose the Bernoulli parameter matrix $\tilde{\mathbf{B}} = \tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top$, it will not improve the model's efficacy as we introduced in the above subsection. That is because the efficiency improvement is mainly from the re-ordering of the computation of the power iteration (i.e., $(\tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top)^k$) which cannot be applied to the power iteration of the sampled adjacency matrix (i.e., $\left(\texttt{sample}(\tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top)\right)^k$).

Mini-Batch. Recall that our optimization objective Eq. (9) includes two kernel matrices $\mathbf{K}_{\mathcal{T}\mathcal{S}}$ and $\mathbf{K}_{\mathcal{S}\mathcal{S}}$. The computation of every entry from $\mathbf{K}_{\mathcal{T}\mathcal{S}}$ and the upper (or lower)-triangle of $\mathbf{K}_{\mathcal{S}\mathcal{S}}$ is independent with each other. In this way, if the computation complexity of every entry from the kernel matrix is $O(C)$, the total complexity of computing $\mathbf{K}_{\mathcal{T}\mathcal{S}}$ and $\mathbf{K}_{\mathcal{S}\mathcal{S}}$ is $O((n_{\mathcal{T}} n_{\mathcal{S}} + \frac{1}{2} n_{\mathcal{S}}^2)C)$, which are resource-intensive for some large graph datasets. Fortunately, KIDD and its variant are easy to be mini-batched. Specifically, at every iteration, a subset of $\mathcal{T}$ and a subset of $\mathcal{S}$ is sampled, and this part of $\mathcal{S}$ is updated by minimizing Eq. (9) through gradient descent. This mini-batch technique is optional and can easily be incorporated with any of the above designs. Our efficiency study experiment shows that the mini-batch is especially important for our proposed KIDD-D and KIDD-LR.

4 EXPERIMENTS

We design extensive experiments to answer the following questions.

- **RQ1.** How effective are the proposed KIDD-D and KIDD-LR compared with the baseline methods?
- **RQ2.** How efficient are the proposed models?

4.1 Experimental Setup

Datasets. In this paper, we select 7 real-world graph classification datasets including NCI1, NCI109, DD, and PROTEINS from TUDataset [37] and ogbg-molhiv, ogbg-molbbbp, and ogbg-molbace from open graph benchmarks [21]. All datasets are publicly available. The detailed dataset statistics are provided in Table 6 in Appendix. For NCI1, NCI109, DD, and PROTEINS, 80/10/10% of the graphs from every dataset are randomly split into the training/validation/test set. For ogbg-molhiv, ogbg-molbbbp, and ogbg-molbace, we use their default dataset split.

Metrics. We select GIN [50] as the downstream GNN, which is trained on the distilled graph dataset $\mathcal{S}$. Its performance on the test graphs is the metric of the corresponding distilled training graphs. To be specific, for NCI1, NCI109, DD, and PROTEINS, accuracy (ACC) is reported and for ogbg-molhiv, ogbg-molbbbp, and ogbg-molbace ROC-AUC is reported as the datasets suggested. We report the average result and the standard deviation in 10 runs.

Baseline methods. We select 4 baseline methods including 3 coreset methods (Random, Herding [47], and K-Center [11, 43]) and a graph dataset distillation method DosCond [26]. Concretely, Random selection is the most naive method which randomly samples $\mathcal{S}$ from $\mathcal{T}$. For Herding and K-Center, we first learn the representation of every training graph by the GIN [50] trained on the whole training set. Then, Herding selects the closest samples to the cluster center of every class. K-Center selects the center samples such that the distance between every node to its nearest center is minimized. Specifically, we implement a greedy solution of K-Center [19] whose initialized set is from Herding. DosCond is a learning-based graph dataset distillation method that matches the training gradient on $\mathcal{S}$ and $\mathcal{T}$ at the initialization step. DosCond is fast, but unlike the exact solution adopted in KIDD, it applies bold approximations for the bi-level distillation objective function.

Implementation of KIDD. In our implementation of KIDD, LiteGNTK is applied to all the graph kernel computations. KIDD-LR is the variant that uses the proposed low-rank speed-up designs and KIDD-D is the variant that uses the Gumbel-Max trick to synthesize discrete graph topology. Mini batch is flexibly applied depending on the size of the target dataset. Detailed parameter settings for reproducibility are provided in Appendix[1].

4.2 Efficacy Study

Effectiveness of KIDD-LR and KIDD-D. For every dataset, 1/10/50 graphs are distilled for every class by baselines and our methods, respectively. After that, the downstream graph classifier GIN [50] is trained on the distilled training graphs, and we report its performance on the test graphs in Table 2. The right-most column shows the downstream classifiers' test performances trained over the entire original training sets. The effectiveness comparison is provided in Table 2. It is observed that

- In most cases, as expected, with the increasing number of training graphs, the downstream graph classifier's performances are improved. This observation is consistent among both the coreset methods (Random, Herding, K-Center) and learning-based distillation methods (DosCond, KIDD-D, and KIDD-LR).

[1]The code is available at https://github.com/pricexu/KIDD .

Table 2: Performance comparison (mean±std). The best and second-best results are bold and underlined, respectively.

Name	Graphs/Cls	Ratio	Random	Herding	K-Center	DosCond	KIDD-D	KIDD-LR	Whole Dataset
NCI1 (ACC)	1	0.06%	57.4±3.0	59.2±3.0	59.2±3.0	57.1±0.9	60.1±0.9	**60.3±1.6**	80.0±1.1
	10	0.61%	59.9±2.0	62.8±0.9	59.1±0.8	60.8±0.9	61.7±1.3	**63.3±1.6**	
	50	3.04%	60.5±2.1	62.5±2.0	59.5±0.5	62.7±0.8	**64.2±0.6**	62.2±0.8	
NCI109 (ACC)	1	0.06%	54.3±2.3	51.7±0.9	51.7±0.9	54.9±2.3	**55.2±2.7**	54.4±1.0	77.7±0.6
	10	0.61%	61.9±1.6	**63.6±0.3**	52.9±1.7	61.4±1.5	63.5±0.5	62.8±0.7	
	50	3.03%	64.0±1.4	64.7±1.2	55.0±2.1	62.9±1.6	**70.4±1.3**	64.7±1.0	
PROTEINS (ACC)	1	0.22%	57.8±1.8	67.6±1.7	67.6±1.7	63.4±1.9	68.8±3.9	**69.3±4.9**	78.6±2.6
	10	2.25%	67.2±0.7	68.3±1.0	71.4±3.3	71.7±0.4	74.1±1.9	**75.3±0.4**	
	50	11.24%	69.6±4.0	70.1±1.0	72.9±2.6	73.2±0.8	75.0±1.9	**75.6±2.3**	
DD (ACC)	1	0.21%	61.3±8.5	60.7±8.4	61.0±3.2	63.0±0.7	65.8±1.7	**69.7±1.0**	76.9±3.2
	10	2.12%	66.8±2.1	67.4±0.7	66.2±2.4	68.1±1.8	70.6±1.3	**71.1±0.8**	
	50	10.62%	71.4±2.2	71.6±1.9	72.3±1.0	70.9±1.0	**73.1±2.2**	71.7±0.7	
ogbg-molhiv (ROC-AUC)	1	<0.01%	0.555±0.036	0.633±0.025	0.633±0.025	0.612±0.025	0.633±0.016	**0.637±0.069**	0.750±0.007
	10	0.06%	0.579±0.012	0.621±0.009	0.630±0.013	0.647±0.031	**0.675±0.054**	0.654±0.019	
	50	0.30%	0.623±0.013	0.616±0.014	0.628±0.014	0.620±0.018	0.708±0.062	**0.709±0.020**	
ogbg-molbbbp (ROC-AUC)	1	0.12%	0.579±0.025	**0.628±0.012**	**0.628±0.012**	0.584±0.030	**0.628±0.011**	0.623±0.006	0.650±0.014
	10	1.23%	0.556±0.003	0.625±0.002	0.596±0.016	0.621±0.013	**0.644±0.011**	0.631±0.015	
	50	6.13%	0.610±0.007	0.630±0.013	0.595±0.018	0.628±0.012	0.662±0.030	**0.663±0.016**	
ogbg-molbace (ROC-AUC)	1	0.17%	0.638±0.009	0.546±0.038	0.546±0.038	0.667±0.021	0.693±0.016	**0.698±0.014**	0.727±0.017
	10	1.65%	0.649±0.017	0.561±0.041	0.658±0.016	0.694±0.018	**0.748±0.020**	0.717±0.012	
	50	8.26%	0.655±0.020	0.703±0.012	0.662±0.013	0.710±0.006	**0.766±0.007**	0.724±0.020	

Table 3: Efficiency comparison (second/iteration) of KIDD with different kernels. $B_{\mathcal{T}}, B_{\mathcal{S}}$ are the batch sizes of the target and synthetic training sets, respectively.

$(B_{\mathcal{T}}, B_{\mathcal{S}})$	Computation	GNTK	LiteGNTK	LiteGNTK-LR
(32, 2)	Forward	0.0475	0.0153	0.0071
	Backprop	0.0241	0.0078	0.0040
(64, 2)	Forward	0.0697	0.0212	0.0111
	Backprop	0.0251	0.0094	0.0045
(128, 16)	Forward	0.2004	0.0551	0.0288
	Backprop	0.1525	0.0433	0.0173
(256, 32)	Forward	OOM	0.1341	0.0745
	Backprop	OOM	0.1450	0.0633

- Interestingly, in some cases (e.g., DD), Random is not always the weakest baseline method, even though it is the most naive one. It reflects that the most representative training samples could be hard to find by heuristics (e.g., Herding, K-Center) and shows the advantages of the learning-based methods (DosCond, KIDD-D, and KIDD-LR).
- The proposed KIDD-D and KIDD-LR obtain the best performance against all the baseline methods under most settings. Strikingly, on ogbg-molbbbp and ogbg-molbace datasets, when the numbers of synthetic graphs are only 1.65%-8.26% of the training graphs, the graph classifiers trained on such tiny datasets are able to outperform the correspondences trained on the complete datasets. It further demonstrates the advantage of learning a representative and informative training set over directly sampling the representative ones from the existing training graphs.

Efficiency study. We first verify the efficiency improvement of our proposed enhanced designs. Specifically, we measure the wall clock time during the forward computation and backward gradient propagation (implemented with PyTorch). 3 models are compared: (1) KIDD-GNTK: using GNTK as the kernel but not our proposed LiteGNTK; (2) KIDD-LiteGNTK: using LiteGNTK as the graph kernel; (3) KIDD-LiteGNTK-LR: using LiteGNTK as the graph kernel and applying the low-rank speed-up technique whose rank is set as 16 in this experiment. All the graph neural network kernels have 5 aggregate operations. For KIDD-GNTK and KIDD-LiteGNTK, they do not include the Gumbel-Max reparameterization trick because it is not proposed for better training efficiency, and in addition, it is an element-wise operation (i.e., $\tilde{\mathbf{A}}[u,v] = \texttt{Gumbel-Max}(\tilde{\mathbf{B}}[u,v])$) whose computation is not heavy. In this experiment, the dataset PROTEINS is used. We test four settings with different batch sizes of the target training graphs $\mathcal{T}$ and the synthetic graphs $\mathcal{S}$. The wall clock time comparison is presented in Table 3 from which we observe that

- KIDD-LiteGNTK is significantly faster (3× faster) than KIDD-GNTK as expected because the non-linearity between aggregation operations is removed, and the aggregation operations (i.e., $\mathbf{A}^k$) can be precomputed. Also, as GNTK contains more operations involving more intermediate variables for modern machine learning packages (e.g., PyTorch) that leads to heavier memory usage. E.g., for the case with batch size $(256, 32)$, KIDD-GNTK is out of memory.
- KIDD-LiteGNTK-LR is much faster than KIDD-LiteGNTK, which aligns well with the Lemma 1.

Next, we compare the wall clock time of KIDD-D and KIDD-LR compared with the learning-based graph dataset distillation

Table 4: Efficiency comparison (second/iteration) with the baseline method DosCond.

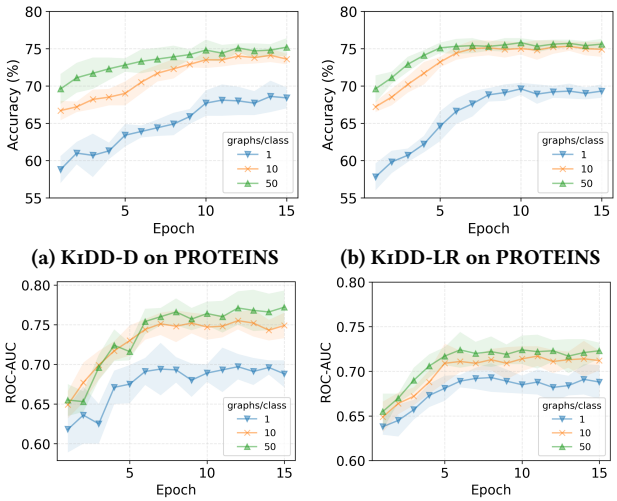
Dataset	Method	Graphs/Class 1	10	50
NCI1	DosCond	0.6755	0.6804	0.7009
	KIDD-D	0.0255	0.0463	0.1540
	KIDD-LR	0.0122	0.0240	0.0826
PROTEINS	DosCond	0.2027	0.2077	0.2112
	KIDD-D	0.0323	0.0608	0.2095
	KIDD-LR	0.0157	0.0313	0.1090
ogbg-molhiv	DosCond	0.0274	0.0311	0.0328
	KIDD-D	0.0281	0.0492	0.1486
	KIDD-LR	0.0135	0.0250	0.0748

method DosCond, which applies a fast approximation of the bilevel gradient matching loss. As the forward and backward computation of DosCond is not clearly defined, whose forward computation of the gradient matching loss involves a gradient backpropagation, we record and compare the total training time for a fixed number of iterations. As the official implementation of Doscond [2] applies mini batch towards the target training set $\mathcal{T}$ while updating $\mathcal{S}$ in a full-batch fashion. For a fair comparison, we follow their settings and set all the methods' batch size of $\mathcal{T}$ as 64, the batch size of $\mathcal{S}$ is the # classes × graphs/class, and both methods' number of the aggregation layers as 3. Then, the wall clock time per update iteration is recorded. The wall clock time comparison is presented in Table 4, from which we observe

- The efficiency of DosCond is consistent with respect to the batch size of the synthetic graphs $\mathcal{S}$. As a comparison, The efficiency of our method KIDD-LR and KIDD-D is more sensitive to the batch size of the synthetic graphs. That is because DosCond's time complexity is linear with respect to $B_{\mathcal{T}} + B_{\mathcal{S}}$ but our method needs to compute two kernel matrices $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ whose numbers of entries are $B_{\mathcal{T}}B_{\mathcal{S}}$ and $B_{\mathcal{S}}^2$, respectively. Here $B_{\mathcal{T}}$, $B_{\mathcal{S}}$ are the batch size of $\mathcal{T}$ and $\mathcal{S}$, respectively.
- As we claimed, both KIDD-D and KIDD-LR cannot scale to very large batch sizes (e.g., 1024). However, with an appropriate batch size (e.g., $B_{\mathcal{T}} = 64$ and $B_{\mathcal{S}} = 20$), two KIDD variants, especially the KIDD-D can have comparable or even better efficiency than DosCond. This is surprising as the DosCond solves the bi-level optimization objective approximately, but KIDD-D provides an exact solution for the distillation optimization objective.

Convergence study. To supplement the efficiency study, a convergence study is provided which shows the models' performance with respect to the increase of training epochs. Here we select the PROTEINS and ogbg-molbace datasets and present the accuracy/ROC-AUC of our proposed models KIDD-D and KIDD-LR in Figure 1a-1d which shows our models can converge quickly within 15 epochs. Notice that in this experiment we select the batch size of $\mathcal{T}$ as 256 indicating on the PROTEINS dataset, 1 epoch is equivalent to 4 update iterations, and on the ogbg-molbace dataset, 1 epoch is equivalent to 7 update iterations. As expected, the training of KIDD-D will be more unstable compared with the training of KIDD-LR due to the sampling operation in the forward computation.

[2] https://github.com/amazon-science/doscond



(a) KIDD-D on PROTEINS (b) KIDD-LR on PROTEINS

(c) KIDD-D on ogbg-molbace (d) KIDD-LR on ogbg-molbace

Figure 1: Model performance vs. training epochs.

Table 5: Ablation study results (mean±std). The metric for the PROTEINS is accuracy and the metric for the ogbg-molbbbp and ogbg-molbace is ROC-AUC. The best is bold.

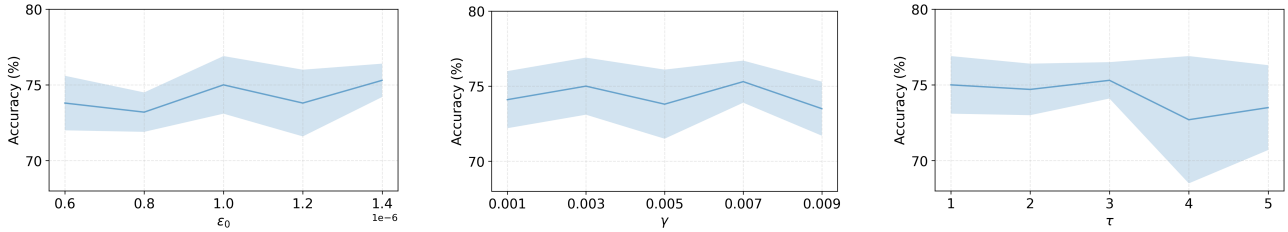
Method	PROTEINS	ogbg-molbbbp	ogbg-molbace
KIDD-RWK	64.8±2.4	0.578±0.009	0.629±0.013
KIDD-LR-NR	72.2±2.8	0.645±0.005	0.699±0.026
KIDD-D-NR	72.0±1.5	0.634±0.030	0.713±0.013
KIDD-LR	**75.6±2.3**	**0.663±0.016**	0.724±0.020
KIDD-D	75.0±1.9	0.662±0.030	**0.766±0.007**

4.3 Auxiliary Experiments

Ablation study. We provide an ablation study to understand the effectiveness of every module of our proposed methods. Specifically, the following models are tested: (1) KIDD-D and KIDD-LR: two variants of our proposed KIDD; (2) KIDD-RWK: using random walk graph kernel (i.e., Eq. (12a), (12c), and (12d)) to compute the kernel matrix; (3) KIDD-D-NR and KIDD-LR-NR: removing the regularization term (i.e., Eq. (10)) from the optimization objective. Datasets PROTEINS, ogbg-molbbbp, and ogbg-molbace are selected and the graphs/class is set as 50. The results are presented in Table 5, from which we observe

- The performance of KIDD-RWK is significantly lower than other KIDD variants equipped with LiteGNTK. It suggests the random walk graph kernel (RWK)-based ridge regression cannot provide generalizable distilled graph datasets for a GNN classifier.
- The regularization term (i.e., Eq. (10)) can improve the performance of KIDD. Including this term to the training loss, our proposed KIDD-LR and KIDD-D obtain the best performances.

Sensitivity Study. There are three main hyperparameters of the proposed KIDD model, ϵ from Eq. (9), γ from Eq. (11), and τ from



(a) Accuracy (%) vs. varying ϵ_0 **(b) Accuracy (%) vs. varying γ** **(c) Accuracy (%) vs. varying initial τ**

Figure 2: Hyperparameter sensitivity studies.

Eq. (17). In this study, we conduct experiments on the PROTEINS dataset with 50 synthetic graphs per class. The KIDD-D variant is tested because the hyperparameter τ is only used for the discrete scenarios. As we mentioned in the experimental settings in Section 4.1, we set $\epsilon = \epsilon_0 \times \frac{\texttt{trace}(\mathbf{K}_{SS})}{n_S}$ so that ϵ is stable with respect to the number of synthetic graphs. Then, we present the model's performance with varying ϵ_0 in Figure 2a. For the different settings of hyperparameter γ, KIDD-D's performance is reported in Figure 2b. For the hyperparameter τ, as mentioned in the hyperparameter settings, we follow the suggestions from [1, 26] and anneal the initial τ to 0.01τ during the training process. Here we compare the performance of KIDD-D with varying initial τ as shown in Figure 2c. In general, we observe that KIDD-D's performance is stable with respect to the selection of hyperparameters ϵ_0 and γ. As τ increases, the model's performance becomes more and more unstable since large τ will lead to large gradients [36] and thus affect the training stability.

5 RELATED WORK

This work lies in the interaction of dataset distillation, graph augmentation, and neural network kernel, as briefly introduced below.

Dataset Distillation. Compressing a large training dataset into a smaller one is a long-standing objective in the machine learning community. A small and informative training dataset can speed up the model tuning and even improve the data privacy [39]. A classic method to compress the training data is named corset which selects training samples from the existing training data. A comprehensive introduction about the coreset is provided by Phillips [42]. Recent progresses [5, 34, 35] formulate the problem under the umbrella of meta-learning (a.k.a. learning to learn). Wang et al. [46] provide an interesting analysis of a simple linear case about dataset distillation. More recently, dataset condensation [57] is proposed whose core idea is to match the training gradient on the distilled dataset and the original one. Another recent attempt is proposed by Cazenavette et al. [5] to match the training trajectories between models trained on the original dataset and on the synthetic dataset.

Besides, the dataset condensation is also grafted into the graph data [27] which shrinks a large training graph into a smaller one for the node-level tasks. The only graph-level dataset distillation method is DosCond [26]. Both DosCond and our work aim to distill the given graph dataset by formulating a bilevel optimization problem whose lower-level problem involves the training dynamics of graph classifier on the synthetic dataset. DosCond speeds up the computation by dropping most of the lower-level training dynamics and only using the training gradient at initialization. Our method achieves higher efficiency by designing the classifier as a GNTK-based KRR whose model's prediction has a closed form.

Graph Data Augmentation. Graph data augmentation [7, 15, 16, 56, 58] aims to learn the given graph(s) such that the performance of the downstream tasks is improved. Our work in this paper can be viewed as augmenting the sampled small set of training graphs from the original training set. The graph augmentation targets can be the graph topology [59], the node features [61], the node labels [4], and the graph labels [40]. Our work focuses on augmenting the graph topology and node features. Due to its flexibility, the graph augmentation techniques have been widely applied to node/edge/graph-level tasks on graph denoising [25, 52], graph imbalanced learning [49, 51, 60], self-supervised learning [53, 54], and many more.

Neural Network Kernel. The relationship between infinitely-wide neural networks and kernel methods are studied for a long time. Early work suggests that one-layer infinitely-wide neural networks with i.i.d. random parameters are equivalent to Gaussian processes [38]. Lee et al. [32] promote the above correspondence into the deep neural network cases. Based on that, recent efforts reveal that gradient descent training of infinitely-wide neural networks with squared loss is identical to the kernel ridge regression [2, 3, 8, 22] (named the neural tangent kernel (NTK)) and similar results are also studied on graph neural networks and named as the graph neural tangent kernel (GNTK) [9]. Our work is an application of the GNTK with our proposed novel enhancements.

6 CONCLUSION

In this paper, we study the graph dataset distillation problem. We propose to exactly solve the bilevel distillation objective through the kernel ridge regression-based method. The graph neural tangent kernel is applied which ensures the distilled dataset can empower the training of the downstream graph neural networks. A series of model enhancements are proposed to ensure the computational tractability of graph neural tangent kernel and in addition, to handle discrete graph topology. Comprehensive experiments are conducted which verify the effectiveness of our proposed models KIDD-D and KIDD-LR and at the same time, show comparable efficiency to the state-of-the-art fast strategies.

ACKNOWLEDGMENTS

ZX and HT are partially supported by NSF (1947135, 2134079 and 1939725), DARPA (HR001121C0165), NIFA (2020-67021-32799), DHS (17STQAC00001-06-00), and ARO (W911NF2110088).

REFERENCES

[1] Abubakar Abid, Muhammad Fatih Balin, and James Zou. 2019. Concrete autoencoders for differentiable feature selection and reconstruction. *arXiv preprint arXiv:1901.09346* (2019).

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. 2019. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*. PMLR, 242–252.

[3] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. 2019. On exact computation with an infinitely wide neural net. *Advances in Neural Information Processing Systems* 32 (2019).

[4] Chen Cai, Dingkang Wang, and Yusu Wang. 2021. Graph Coarsening with Neural Networks. In *International Conference on Learning Representations*.

[5] George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. 2022. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4750–4759.

[6] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2005. Bilevel programming: A survey. *4or* 3, 2 (2005), 87–107.

[7] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. 2022. Data augmentation for deep graph learning: A survey. *arXiv preprint arXiv:2202.08235* (2022).

[8] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. 2019. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*. PMLR, 1675–1685.

[9] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. 2019. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems* 32 (2019).

[10] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. 2020. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 169–177.

[11] Reza Zanjirani Farahani and Masoud Hekmatfar. 2009. *Facility location: concepts, models, algorithms and case studies*. Springer Science & Business Media.

[12] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. 2017. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*. PMLR, 1165–1173.

[13] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. 2018. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*. PMLR, 1568–1577.

[14] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning discrete structures for graph neural networks. In *International conference on machine learning*. PMLR, 1972–1982.

[15] Dongqi Fu and Jingrui He. 2022. Natural and Artificial Dynamics in Graphs: Concept, Progress, and Future. *Frontiers Big Data* 5 (2022). https://doi.org/10.3389/fdata.2022.1062637

[16] Dongqi Fu, Zhe Xu, Hanghang Tong, and Jingrui He. 2023. Natural and Artificial Dynamics in GNNs: A Tutorial. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, WSDM 2023, Singapore, 27 February 2023 - 3 March 2023*, Tat-Seng Chua, Hady W. Lauw, Luo Si, Evimaria Terzi, and Panayiotis Tsaparas (Eds.). ACM, 1252–1255. https://doi.org/10.1145/3539597.3572726

[17] Kunihiko Fukushima. 1975. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics* 20, 3 (1975), 121–136.

[18] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*.

[19] Teofilo F Gonzalez. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical computer science* 38 (1985), 293–306.

[20] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 895–904.

[21] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[22] Arthur Jacot, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems* 31 (2018).

[23] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rkE3y85ee

[24] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. 2021. Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models. *Journal of cheminformatics* 13, 1 (2021), 1–23.

[25] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 66–74.

[26] Wei Jin, Xianfeng Tang, Haoming Jiang, Zheng Li, Danqing Zhang, Jiliang Tang, and Bing Yin. 2022. Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 720–730.

[27] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Graph Condensation for Graph Neural Networks. In *International Conference on Learning Representations*.

[28] U Kang, Hanghang Tong, and Jimeng Sun. 2012. Fast random walk graph kernel. In *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 828–838.

[29] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. 2003. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 321–328.

[30] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[31] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[32] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. 2018. Deep Neural Networks as Gaussian Processes. In *International Conference on Learning Representations*.

[33] Liangyue Li, Yuan Yao, Jie Tang, Wei Fan, and Hanghang Tong. 2016. Quint: on query-specific optimal networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 985–994.

[34] Jonathan Lorraine, Paul Vicol, and David Duvenaud. 2020. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1540–1552.

[35] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*. PMLR, 2113–2122.

[36] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=S1jE5L5gl

[37] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. arXiv:2007.08663 www.graphlearning.io

[38] Radford M Neal. 1996. Priors for infinite networks. In *Bayesian Learning for Neural Networks*. Springer, 29–53.

[39] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. 2021. Dataset Meta-Learning from Kernel Ridge-Regression. In *International Conference on Learning Representations*.

[40] Joonhyung Park, Hajin Shim, and Eunho Yang. 2022. Graph transplant: Node saliency-guided graph mixup with local structure preservation. In *Proceedings of the First MiniCon Conference*.

[41] Fabian Pedregosa. 2016. Hyperparameter optimization with approximate gradient. In *International conference on machine learning*. PMLR, 737–746.

[42] Jeff M Phillips. 2017. Coresets and sketches. In *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 1269–1288.

[43] Ozan Sener and Silvio Savarese. 2018. Active Learning for Convolutional Neural Networks: A Core-Set Approach. In *International Conference on Learning Representations*.

[44] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. 2019. Truncated back-propagation for bilevel optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 1723–1732.

[45] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. 2021. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems* 34 (2021), 15920–15933.

[46] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. 2018. Dataset distillation. *arXiv preprint arXiv:1811.10959* (2018).

[47] Max Welling. 2009. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 1121–1128.

[48] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[49] Lirong Wu, Haitao Lin, Zhangyang Gao, Cheng Tan, Stan Li, et al. 2021. Graphmixup: Improving class-imbalanced node classification on graphs by self-supervised context prediction. *arXiv preprint arXiv:2106.11133* (2021).

[50] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[51] Zhe Xu, Kaize Ding, Yu-Xiong Wang, Huan Liu, and Hanghang Tong. 2022. Generalized Few-Shot Node Classification. In *IEEE International Conference on Data Mining, ICDM 2022, Orlando, FL, USA, November 28 - Dec. 1, 2022*, Xingquan Zhu, Sanjay Ranka, My T. Thai, Takashi Washio, and Xindong Wu (Eds.). IEEE, 608–617. https://doi.org/10.1109/ICDM54844.2022.00071

[52] Zhe Xu, Boxin Du, and Hanghang Tong. 2022. Graph sanitation with application to node classification. In *Proceedings of the ACM Web Conference 2022*. 1136–1147.

[53] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph contrastive learning automated. In *International Conference on Machine Learning*. PMLR, 12121–12132.

[54] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems* 33 (2020), 5812–5823.

[55] Si Zhang, Dawei Zhou, Mehmet Yigit Yildirim, Scott Alcorn, Jingrui He, Hasan Davulcu, and Hanghang Tong. 2017. Hidden: hierarchical dense subgraph detection with application to financial fraud detection. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 570–578.

[56] Beidi Zhao, Boxin Du, Zhe Xu, Liangyue Li, and Hanghang Tong. 2022. Learning Optimal Propagation for Graph Neural Networks. *arXiv preprint arXiv:2205.02998* (2022).

[57] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2021. Dataset Condensation with Gradient Matching. *ICLR* 1, 2 (2021), 3.

[58] Tong Zhao, Gang Liu, Stephan Günnemann, and Meng Jiang. 2022. Graph Data Augmentation for Graph Machine Learning: A Survey. *arXiv preprint arXiv:2202.08871* (2022).

[59] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11015–11023.

[60] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM international conference on web search and data mining*. 833–841.

[61] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*. 2069–2080.

A EXACT COMPUTATION OF THE UPDATE FUNCTION

To compute the Eq. (4b) and Eq. (4c) exactly, according to [3, 9], the following properties of the ReLU activation function should be used. If $\tilde{\mathbf{\Lambda}} = \begin{pmatrix} 1 & \lambda \\ \lambda & 1 \end{pmatrix}$,

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\tilde{\mathbf{\Lambda}})}\sigma(a)\sigma(b) = \frac{\lambda(\pi - \arccos(\lambda)) + \sqrt{1-\lambda^2}}{2\pi}, \tag{18a}$$

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\tilde{\mathbf{\Lambda}})}\dot{\sigma}(a)\dot{\sigma}(b) = \frac{\pi - \arccos(\lambda)}{2\pi}. \tag{18b}$$

Also, using the homogeneity of ReLU (i.e., for $\forall a \geq 0$, $\sigma(ax) = a\sigma(x)$, and $\dot{\sigma}(ax) = \dot{\sigma}(x)$ we can decompose the $\mathbf{\Lambda}_{\mathcal{G},\mathcal{G}'}[u,u'] = \mathbf{D}\tilde{\mathbf{\Lambda}}\mathbf{D}$, where $\mathbf{D} = \begin{pmatrix} c_1 & 0 \\ 0 & c_2 \end{pmatrix}$ and get

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\mathbf{D}\tilde{\mathbf{\Lambda}}\mathbf{D})}\sigma(a)\sigma(b) = c_1c_2\frac{\lambda(\pi - \arccos(\lambda)) + \sqrt{1-\lambda^2}}{2\pi}, \tag{19a}$$

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\mathbf{D}\tilde{\mathbf{\Lambda}}\mathbf{D})}\dot{\sigma}(a)\dot{\sigma}(b) = \frac{\pi - \arccos(\lambda)}{2\pi}. \tag{19b}$$

B PROOF OF LEMMA 1

Lemma 1. (Time Complexity) Assume both $\mathcal{G}$ and $\tilde{\mathcal{G}}$ have n nodes, i.e., $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \in \mathbb{R}^{n\times n}$, the time complexity of computing $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k$ is $O(rn^2)$. The time complexity of computing $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{U}}}$ and $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{V}}}$ is $O(r^3n^3)$.

Proof. $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k$ can be rewritten as $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\tilde{\mathbf{U}}^\top$. Thus, the complexity of computing $(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}$ is $O(rn^2 + (k-2)r^3)$, the complexity of multiply $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}$, $\tilde{\mathbf{V}}$, $(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}$, $\tilde{\mathbf{U}}^\top$ from the left-hand side is $O(2rn^2 + r^2n)$. Hence, putting everything together the time complexity of computing $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k$ is $O(3rn^2 + r^2n + (k-2)r^3)$ which can be shortened as $O(rn^2)$ given $r \ll n$ and $k \ll n$.

$\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{U}}} = \frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\tilde{\mathbf{U}}^\top}{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}\frac{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}{\partial\tilde{\mathbf{U}}} + \frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\hat{\mathbf{U}}^\top}{\partial\hat{\mathbf{U}}}$, where $\hat{\mathbf{U}} = \tilde{\mathbf{U}}$. It is a fact that, for any matrix $\mathbf{M} \in \mathbb{R}^{r\times r}$

$$\begin{aligned}\frac{\partial\mathbf{M}^k[i,j]}{\partial\mathbf{M}[s,t]} &= \frac{\partial\sum_{l_1,\ldots,l_{k-1}}\mathbf{M}[i,l_1]\mathbf{M}[l_1,l_2]\ldots\mathbf{M}[l_{k-1},j]}{\partial\mathbf{M}[s,t]}, \\ &= \Big(\sum_{l=0}^{k-1}\mathbf{M}^l\otimes(\mathbf{M}^\top)^{k-1-l}\Big)[ri+j, rs+t].\end{aligned} \tag{20}$$

Thus, if we represent $\frac{\partial(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}}{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}$ in the shape of $\mathbb{R}^{r^2\times r^2}$ it can be computed as $\sum_{l=0}^{k-2}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^l\otimes(\tilde{\mathbf{V}}^\top\tilde{\mathbf{U}})^{k-2-l}$ whose time complexity is $O((k-1)(2r^2n + r^4))$. The time complexity of computing $\frac{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}{\partial\tilde{\mathbf{U}}}$ is $O(r^3n)$. The complexity of multiply $\frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\tilde{\mathbf{U}}^\top}{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}$ and $\frac{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}{\partial\tilde{\mathbf{U}}}$ is $O(r^3n^3)$. The time complexity of computing $\frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\tilde{\mathbf{U}}^\top}{\partial\hat{\mathbf{U}}}$ is $O(rn^3)$. Consequently, put everything together, the time complexity of computing $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{U}}}$ is $O(r^3n^3)$ given $r \ll n$ and $k \ll n$.

The time complexity of computing $\frac{\partial\mathbf{\Theta}_{\tilde{\mathcal{G}},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{V}}}$ can be analyzed similarly and is omitted for brevity. □

C PROOF OF LEMMA 2

Lemma 2. (Error Bound) Given a target graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$, a synthetic graph $\tilde{\mathcal{G}} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\}$ and $\tilde{\mathcal{G}}$'s rank-r representation $\tilde{\mathcal{G}}_r = \{\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top, \tilde{\mathbf{X}}\}$, if we assume both $\mathcal{G}$ and $\tilde{\mathcal{G}}$ have n nodes, $\mathcal{G}$ has m edges, and $\mathbf{c}_{\mathcal{G}} = \mathbf{c}_{\tilde{\mathcal{G}}} = \mathbf{c}_{\tilde{\mathcal{G}}_r} = \mathbf{1}$ are all-one vectors, the error of the LiteGNTK value after applying the low-rank speed-up can be bounded by

$$\mathbf{K}_{\texttt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}) - \mathbf{K}_{\texttt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}_r) \leq nm^{\frac{k}{2}}||\mathbf{\Theta}||_F\sum_{i=r+1}^{n}|\tilde{\lambda}_i^k|, \tag{21}$$

where $\mathbf{\Theta} = \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} = \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}_r}$ is the output of Eq. (12b) and $\tilde{\lambda}_i$ is the i-th largest eigenvalue of $\tilde{\mathbf{A}}$.

Proof. The difference between $\mathbf{K}_{\texttt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}})$ and $\mathbf{K}_{\texttt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}_r)$ can be presented as

$$\begin{aligned}&|\mathbf{c}_{\tilde{\mathcal{G}}}^\top\mathbf{A}^k\mathbf{\Theta}(\tilde{\mathbf{A}}^k - (\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top)^k)\mathbf{c}_{\mathcal{G}}| \\ &= |\mathbf{c}_{\tilde{\mathcal{G}}}^\top\mathbf{A}^k\mathbf{\Theta}(\sum_{i=r+1}^{n}\tilde{\lambda}_i^k\tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top)\mathbf{c}_{\mathcal{G}}| \\ &\leq ||\mathbf{c}_{\mathcal{G}}||_2||\mathbf{c}_{\tilde{\mathcal{G}}}||_2||\mathbf{A}||_F^k||\mathbf{\Theta}||_F||\sum_{i=r+1}^{n}\tilde{\lambda}_i^k\tilde{\mathbf{u}}_i\tilde{\mathbf{u}}_i^\top||_F \\ &\leq nm^{\frac{k}{2}}||\mathbf{\Theta}||_F\sum_{i=r+1}^{n}|\tilde{\lambda}_i^k|\end{aligned} \tag{22}$$

where $\tilde{\mathbf{u}}_i$ is the i-th unit eigenvector of $\tilde{\mathbf{A}}$, and the last inequality holds because (1) $\{\mathbf{u}_i\}$ are the normalized eigenvectors, and (2) $||\sum_i a_i\mathbf{u}_i\mathbf{u}_i^\top||_F = \sqrt{\texttt{trace}(\sum_i a_i^2\mathbf{u}_i\mathbf{u}_i^\top)} = \sqrt{\sum_i a_i^2\texttt{trace}(\mathbf{u}_i\mathbf{u}_i^\top)} \leq$

Algorithm 1: KIDD-LR

Input : a target graph dataset $\mathcal{T} = \{(\mathcal{G}_i, y_i)\}_0^{n_\mathcal{T}-1}$, the size of the synthetic dataset $n_\mathcal{S}$;
Output : the synthetic dataset $\mathcal{S}$;

initialization: sample $n_\mathcal{S}$ graphs and their labels y_i from the $\mathcal{T}$ as the initial $\mathcal{S}$; decompose adjacency matrices into their low-rank matrices (e.g., $\tilde{\mathbf{U}}_i, \tilde{\mathbf{V}}_i$) by SVD.
while $\mathcal{S}$ *is not converged* **do**
  compute $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ by Eq. (12a), (12b), Eq. (15), and Eq. (12d);
  Update $\mathcal{S} = \{\tilde{\mathbf{U}}_i, \tilde{\mathbf{V}}_i, \tilde{\mathbf{X}}_i\}_0^{n_\mathcal{S}-1}$ based on the gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{S}}$ from Eq. (9);
end
clip $\{\tilde{\mathbf{A}}_i = \tilde{\mathbf{U}}_i \tilde{\mathbf{V}}_i^\top\}_0^{n_\mathcal{S}-1}$ to be non-negative;
return $\mathcal{S} = \{\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i, y_i\}_0^{n_\mathcal{S}-1}$.

Algorithm 2: KIDD-D

Input : a target graph dataset $\mathcal{T} = \{(\mathcal{G}_i, y_i)\}_0^{n_\mathcal{T}-1}$, the size of the synthetic dataset $n_\mathcal{S}$;
Output : the synthetic dataset $\mathcal{S}$;

initialization: sample $n_\mathcal{S}$ graphs and their labels y_i from the $\mathcal{T}$ as the initial $\mathcal{S}$; initialize the Bernoulli parameter matrix $\tilde{\mathbf{B}}_i$ of every synthetic graph according to their adjacency matrix: $\tilde{\mathbf{B}}_i[u, v] = C$ if $\tilde{\mathbf{A}}_i[u, v] = 1$; otherwise $\tilde{\mathbf{B}}_i[u, v] = -C$, where C is a constant, e.g., 1.
while $\mathcal{S}$ *is not converged* **do**
  Sample adjacency matrices of synthetic graphs $\{\tilde{\mathbf{A}}_i\}_0^{n_\mathcal{S}-1}$ by Eq. (17) and $\{\tilde{\mathbf{B}}_i\}_0^{n_\mathcal{S}-1}$;
  compute $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ by Eq. (12a)-(12d);
  Update $\mathcal{S} = \{\tilde{\mathbf{B}}_i, \tilde{\mathbf{X}}_i\}_0^{n_\mathcal{S}-1}$ based on the gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{S}}$ from Eq. (9);
end
Discretize the adjacency matrix of every synthetic graph: $\tilde{\mathbf{A}}_i[u, v] = 1$ if $\texttt{sigmoid}(\tilde{\mathbf{B}}_i[u, v]) > 0.5$, else $\tilde{\mathbf{A}}_i[u, v] = 0$;
return $\mathcal{S} = \{\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i, y_i\}_0^{n_\mathcal{S}-1}$.

$\sum_i |a_i|$. The above bound can be further simplified by limiting $\tilde{\mathbf{X}}$, $\mathbf{X}$, and $\mathbf{\Theta}$ to special cases but we keep this form for generality. □

D DATASET STATISTICS

The detailed dataset statistics are provided in Table 6.

Table 6: Dataset statistics.

Name	# Graphs	# Nodes	# Edges	# Features	# Classes
NCI1	4110	29.9	32.3	37	2
NCI109	4127	29.7	32.1	38	2
PROTEINS	1113	39.1	72.8	4	2
DD	1178	284.3	715.7	89	2
ogbg-molhiv	41127	25.5	54.9	9	2
ogbg-molbbbp	2039	24.1	51.9	9	2
ogbg-molbace	1513	34.1	73.7	9	2

E REPRODUCIBILITY

Hardwares. We implement KIDD-LR and KIDD-D in PyTorch[3] and PyTorch-geometric[4]. All the efficiency study results are from one NVIDIA Tesla V100 SXM2-32GB GPU on a server with 96 Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz processors and 1.5T RAM.
Parameter Settings. The parameters of the KIDD-D and KIDD-LR are set as follows. The node scaling factor c_u is set as 1 for every node u from $\mathcal{T}$ and $\mathcal{S}$. The learning rate of KIDD-D and KIDD-LR is searched in $\{1e-1, 1e-2, 1e-3\}$. The ϵ is set as $\epsilon = \epsilon_0 \times \frac{\texttt{trace}(\mathbf{K}_{\mathcal{SS}})}{n_\mathcal{S}}$ so that it is stable with respect to the size of $\mathbf{K}_{\mathcal{SS}}$ and the ϵ_0 is set as $1e-6$. γ is searched in $\{0, 1e-4, 1e-3, 1e-2\}$. For KIDD-D, the τ is annealed during the training as [1] suggested. The initial τ is set as 1 and annealed to 0.01 at and after epoch 100. The rank r of the KIDD-LR is searched in $\{16, 32\}$. We plan to release the code upon publication.

F DETAILED ALGORITHMS

Detailed algorithms of KIDD-LR and KIDD-D are provided in Algorithm 1 and Algorithm 2, respectively.

G LIMITATION AND FUTURE WORK

The main limitation of the KIDD-LR and KIDD-D, as we claimed in Section 4.2, lies in its computation complexity scales with respect to $(B_\mathcal{T} B_\mathcal{S} + B_\mathcal{S}^2)$ where $B_\mathcal{T}$ and $B_\mathcal{S}$ are the batch sizes of the target training set and the synthetic set. In other words, KIDD-LR and KIDD-D are limited to medium batch sizes (e.g., $B_\mathcal{T} = 256$ and $B_\mathcal{S} = 64$). Although such batch sizes can handle most real-world graph datasets, improving the scalability of KIDD-LR and KIDD-D is interesting and worth studying.

[3] https://pytorch.org/
[4] https://pytorch-geometric.readthedocs.io/en/latest/