# Data-Centric Graph Condensation via Diffusion Matching

**Editors:** Hung-yi Lee and Tongliang Liu

## Abstract

This paper introduces Data-Centric Graph Condensation (named DCGC), a task- and model-agnostic method for condensing a large graph into a smaller one by matching the distribution between two graphs. DCGC defines the distribution of a graph as the trajectories of its node signals (such as node features and node labels) induced by a diffusion process over the geometric structure, which accommodates multi-order structural information. Built upon this, DCGC compresses the topological knowledge of the original graph into the orders-of-magnitude smaller synthetic one by aligning their distributions in input space. Compared with existing methods that stick to particular GNN architectures and require solving complicated optimization, DCGC can be flexibly applied to arbitrary off-the-shelf GNNs and achieve graph condensation with a much faster speed. Apart from the cross-architecture generalization ability and training efficiency, experiments demonstrate that DCGC yields consistently superior performance than existing methods on datasets with varying scales and condensation ratios.

**Keywords:** Dataset distillation, Graph Condensation, Data-oriented methods.

## 1. Introduction

Graphs are a generic representation for systems of certain interactions and structures, such as large online social networks (Fan et al., 2019), user-item recommender systems (Wu et al., 2019), chemical molecules (Stärk et al., 2022), and biological protein interactions (Réau et al., 2023). Recent advances in deep learning-based methods on graph-structured data, such as graph neural networks (Kipf and Welling, 2017; Velickovic et al., 2018), have garnered significant attention and research interest. However, training deep graph networks on large real-world graphs requires tremendous computational and infrastructural resources due to the necessity of performing message passing layer by layer among inter-connected nodes (Zeng et al., 2020).

To address this challenge, a natural idea is to compress the dataset involving data structures, which, in particular, entails reducing the number of nodes and edges in the graph. To this end, traditional methods include graph sparsification (Spielman and Teng, 2011) and graph coarsening (Loukas and Vandergheynst, 2018; Cai et al., 2021; Kumar et al., 2023): the former aims to obtain a sparser graph by removing edges from the original graph, while the latter targets reducing the number of nodes by extracting a subset from the node-set. However, these methods often rely on some predefined heuristics and lack guidance from training (Yang et al., 2023), making it difficult to achieve satisfactory results on downstream tasks.

**Existing Works.** Another technical path showing empirical success in recent studies resorts to a synthesis-based approach that directly learns the node feature matrix and the adjacency matrix of the target compressed graph (a.k.a. synthetic graph), which is called

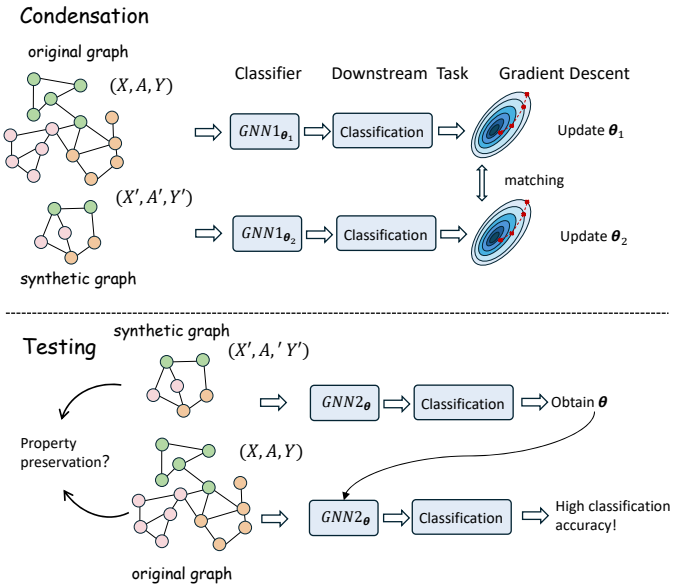


Figure 1: Limitations of existing gradient matching methods. Top: Gradient matching is supervised, task-oriented and time-consuming. The success of gradient matching relies on an accurate matching of the gradient update trajectories. Bottom: In testing, another GNN has to be trained on the synthetic graph for evaluation. The differences between the GNNs used in condensation/testing might cause cross-architecture generalization issues.

*graph condensation* or *graph distillation* (Jin et al., 2022; Liu et al., 2022; Yang et al., 2023; Zheng et al., 2023; Zhang et al., 2024; Fang et al., 2024; Liu et al., 2024). These methods share a similar spirit, aiming to learn a synthetic graph that can replicate the same gradient trajectory of model parameters as the original graph, named gradient-matching (Zhao et al., 2021). Although these methods have achieved promising performance, their design philosophies lead to unsatisfactory capabilities (Gupta et al., 2024) due to failure to the following reasons:

- **Task- and Model-Oriented Nature.** Gradient matching is inherently task-oriented and model-oriented, as it requires a specific downstream task (e.g., node classification) and a pre-defined GNN architecture to guide the condensation process. This reliance on task-specific supervision (e.g., node labels) limits its applicability to unsupervised or semi-supervised scenarios where labeled data may be scarce or unavailable. Furthermore, the synthetic graph generated through this process is tailored to a specific task and model, which may not generalize well to other tasks or architectures.

- **Lack of Graph Property Preservation.** While gradient matching ensures that the GNN trained on the synthetic graph achieves similar performance on the downstream task, it does not guarantee that the synthetic graph exhibits structural or attribute properties similar to those of the original graph. For example, the synthetic graph

may fail to preserve important global or local topological features, such as community structure or node degree distribution. Existing work often overlooks this critical aspect, focusing solely on task performance without evaluating whether the synthetic graph faithfully captures the intrinsic properties of the original graph.

- **Computational Inefficiency.** The primary motivation for graph condensation is to improve computational efficiency by reducing the size of the graph. However, gradient matching-based methods are themselves computationally expensive. They require training a GNN on the original graph, which can be time-consuming for large graphs, and involve a complex bi-level optimization process to match gradients during the condensation phase. This overhead undermines the goal of efficiency, making the approach less practical for real-world applications where scalability is a key concern.

**Presented Work.** To address these limitations, this paper proposes **D**ata-**C**entric **G**raph **C**ondensation via Diffusion Matching (DCGC in short). DCGC inherits the spirit of distribution matching (Zhao and Bilen, 2023), learning the condensed graph by minimizing the divergence between the distributions of the original graph and the synthetic graph. Observing that a graph is a mixture of the node signals (e.g., node features and labels) and their connections, we seek a principled means to characterize and extract the topological knowledge from the original graph entangled with these node signals. In particular, we resort to an analogy between a geometric diffusion process that updates node signals through time and a non-parametric propagation on graphs that returns aggregated node features at different layers. On top of this, we decompose a graph into a collection of node signals, where each node's signal is aggregated from its multi-order structural information, and the distribution of a graph is subsequently defined as the distribution of the aggregated node signals. The divergence between the original graph and the synthetic graph is further measured by the Maximum Mean Discrepancy, which can be easily optimized in linear time w.r.t. the graph size.

DCGC addresses the limitations of the above works in the following ways. Unlike gradient matching, which is task- and model-oriented, our approach is **task-agnostic and model-agnostic**, as it relies solely on the intrinsic properties of the graph captured by the diffusion process, eliminating the need for supervised labels or a predefined GNN architecture. By matching the diffusion trajectories of node features and labels (if available), our method ensures that the synthetic graph preserves both **local and global structural properties** of the original graph, addressing the lack of graph property preservation in gradient matching. Furthermore, our approach avoids the computational inefficiency of gradient matching by using efficient discrete approximations (e.g., Euler's method) to simulate the diffusion process, making it scalable to large graphs. Finally, since the diffusion process is independent of any specific GNN architecture, the synthetic graphs generated by our method exhibit strong **cross-architecture generalization**, ensuring consistent performance across different models.

We evaluate the proposed DCGC on eight graph datasets of varying scales and properties. The experimental results demonstrate that the synthetic graphs condensed by DCGC can preserve the important graph properties in both unsupervised and supervised settings. In terms of the utilities, graphs condensed by DCGC yield comparable or even better performance than existing SOTA gradient-matching methods. In cross-architecture settings and

on heterophilic datasets, DCGC exhibits superior and more stable performance across different GNN architectures. Specifically, apart from improving the averaged accuracy, DCGC reduces the cross-architecture standard deviation by an average of 26.3%. In terms of training speed, compared to the current fastest gradient matching methods, DCGC reduces the training time by 96.4%. These results clearly demonstrate the superiority of DCGC in terms of efficacy, generalization ability, and efficiency.

## 2. Preliminaries

**Graph Notations.** We define a graph as $\mathcal{G} = \{\mathbf{X}, \mathbf{A}\}$, which consists of a node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ for $N$ nodes and a corresponding adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. The primary objective of graph condensation is to synthesize a much smaller graph, denoted as $\mathcal{S} = \{\mathbf{X}', \mathbf{A}'\}$, with $N'$ nodes (where $N' \ll N$). This synthetic graph is optimized to encapsulate the essential properties and statistical information of the original large graph $\mathcal{G}$.

**Supervised Setting.** This work addresses graph condensation under a supervised setting. In the *supervised setting*, the supervisory information is provided in the form of a one-hot node label matrix $\mathbf{Y} \in \mathbb{R}^{N \times C}$, where $C$ is the total number of classes. We use $N_c$ and $N'_c$ to denote the number of nodes belonging to class $c$ within the original graph $\mathcal{G}$ and the synthetic graph $\mathcal{S}$, respectively.

**Maximum Mean Discrepancy.** A pivotal tool for measuring the divergence between two probability distributions is the Maximum Mean Discrepancy (Gretton et al., 2012) (MMD). Given two distributions, $\mathbb{X}$ and $\mathbb{Y}$, the MMD is defined as the largest difference in expectations over functions within the unit ball of a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$. Formally, it is expressed as:

$$\mathsf{MMD}(\mathbb{X}, \mathbb{Y}) = \sup_{\|f\|_{\mathcal{H}} \leq 1} \left(\mathbb{E}_{\boldsymbol{x} \sim \mathbb{X}}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{y} \sim \mathbb{Y}}[f(\boldsymbol{y})]\right) = \|\mu_{\mathbb{X}} - \mu_{\mathbb{Y}}\|_{\mathcal{H}}, \tag{1}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are samples drawn from $\mathbb{X}$ and $\mathbb{Y}$, respectively. The terms $\mu_{\mathbb{X}} = \mathbb{E}_{\mathbb{X}}[\phi(\boldsymbol{x})]$ and $\mu_{\mathbb{Y}} = \mathbb{E}_{\mathbb{Y}}[\phi(\boldsymbol{y})]$ represent the mean embeddings of the distributions in the RKHS, with $\phi(\cdot)$ being the feature map associated with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ such that $f(\cdot) = \langle f, \phi(\cdot) \rangle_{\mathcal{H}}$. In practice, minimizing the MMD is achieved by minimizing its squared value, which can be computed efficiently using the kernel trick:

$$\begin{aligned}
\mathsf{MMD}^2(\mathbb{X}, \mathbb{Y}) &= \langle \mu_{\mathbb{X}} - \mu_{\mathbb{Y}}, \mu_{\mathbb{X}} - \mu_{\mathbb{Y}} \rangle_{\mathcal{H}} \\
&= \langle \mu_{\mathbb{X}}, \mu_{\mathbb{X}} \rangle_{\mathcal{H}} + \langle \mu_{\mathbb{Y}}, \mu_{\mathbb{Y}} \rangle_{\mathcal{H}} - 2 \langle \mu_{\mathbb{X}}, \mu_{\mathbb{Y}} \rangle_{\mathcal{H}} \\
&= \mathbb{E}_{\boldsymbol{x}, \boldsymbol{x}' \sim \mathbb{X}} \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle_{\mathcal{H}} + \mathbb{E}_{\boldsymbol{y}, \boldsymbol{y}' \sim \mathbb{Y}} \langle \phi(\boldsymbol{y}), \phi(\boldsymbol{y}') \rangle_{\mathcal{H}} - 2\, \mathbb{E}_{\boldsymbol{x} \sim \mathbb{X}, \boldsymbol{y} \sim \mathbb{Y}} \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{H}} \\
&= \mathbb{E}_{\boldsymbol{x}, \boldsymbol{x}' \sim \mathbb{X}}[\kappa(\boldsymbol{x}, \boldsymbol{x}')] + \mathbb{E}_{\boldsymbol{y}, \boldsymbol{y}' \sim \mathbb{Y}}[\kappa(\boldsymbol{y}, \boldsymbol{y}')] - 2\, \mathbb{E}_{\boldsymbol{x} \sim \mathbb{X}, \boldsymbol{y} \sim \mathbb{Y}}[\kappa(\boldsymbol{x}, \boldsymbol{y})],
\end{aligned} \tag{2}$$

where $\kappa(\cdot, \cdot)$ is the kernel function associated with the RKHS $\mathcal{H}$. A common choice is the Gaussian kernel, defined as $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|^2 / (2\sigma^2))$, where $\sigma$ is the bandwidth parameter.

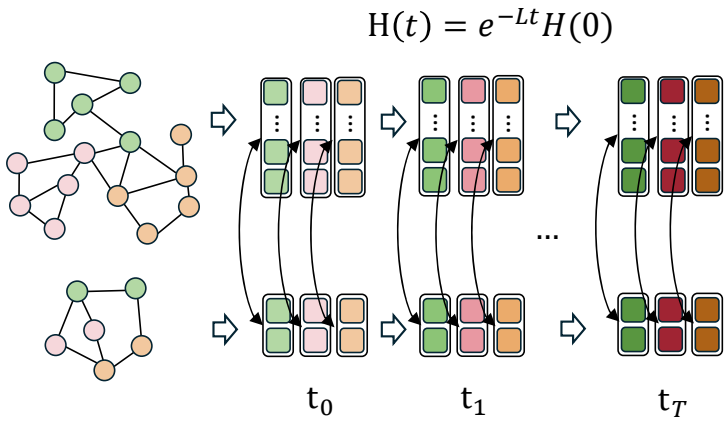


Figure 2: Illustration of graph diffusion process and diffusion matching. The distribution of a graph is defined as the diffusion trajectories of the node signal (e.g., node features/labels).

# 3. Methodology

## 3.1. Condensation via Distribution Matching

To compress a graph into a synthetic one such that the graph properties are preserved, a natural approach is to ensure that the distribution of synthetic data closely resembles the distribution of real data (Zhao and Bilen, 2023). If we denote the distribution of the original graph $\mathcal{G}$ by $\mathbb{G}$, then, distribution matching minimizes the discrepancy between $\mathbb{G}$ and $\mathbb{S}$:

$$\min_{\mathcal{S}=(\mathbf{X}',\mathbf{A}')} \mathcal{D}(\mathbb{G}, \mathbb{S}) \tag{3}$$

where $\mathcal{D}$ is a measure of the divergence between two distributions. When the distribution of a graph, as well as the difference between the two distributions, is appropriately defined, we are able to solve the graph condensation problem by solving the above optimization problems. However, since graph data is non-i.i.d., it remains a challenge how to define the distribution between two graphs and how to quantify their divergence properly.

## 3.2. Node-wise Diffusion Trajectories as a Measurement of Graph Distribution

Defining a meaningful distribution for graph data is inherently challenging due to the complex and non-Euclidean nature of graphs. Unlike traditional data types such as images or text, where samples are often assumed to be independent and identically distributed (i.i.d.), graph data exhibits strong dependencies between node features and the underlying graph structure. Specifically, the features of a node are typically influenced by its neighbors through edges, making node features non-i.i.d. and highly structured. This interdependence necessitates a joint modeling of both node attributes and graph topology, which is non-trivial due to the combinatorial nature of graphs. Additionally, graphs can vary significantly in size, connectivity patterns, and node permutations, further complicating the

definition of a consistent and generalizable distribution. These challenges render traditional distribution matching techniques, which rely on explicit probability density functions or i.i.d. assumptions (Zhao and Bilen, 2023), inadequate for graph-structured data.

#### 3.2.1. Node signal distribution with graph structure

To resolve this challenge, we resort to graph diffusion process (Kondor and Vert, 2004; Wang et al., 2021; Wu et al., 2023), which utilizes the diffusion ODE to characterize the evolution of a graph signal (e.g., node features) under the spatial constraint of the graph structure. Formally, let $\mathbf{H}$ be the node-wise graph signal, the diffusion process of signal $\mathbf{H}$ (e.g., the node feature matrix $\mathbf{X}$) along the graph structure is:

$$\frac{\mathrm{d}\mathbf{H}(t)}{\mathrm{d}t} = -\mathbf{K}\mathbf{H}(t),\ \mathbf{H}(0) = \mathbf{H}, \tag{ODE}$$

$$\mathbf{H}(t) = \exp(-\mathbf{K}t)\ \mathbf{H}(0), \tag{solution}$$

where $\mathbf{H}(t)$ is the corresponding node signal matrix at time $t$, and $\mathbf{K} \in \mathbb{R}^{N\times N}$ is the generalized diffusion kernel. One representative diffusion kernel is the heat kernel (Kondor and Vert, 2004) with the following specification $\mathbf{K} = \mathbf{L}$, where $\mathbf{L}$ is the (normalized) graph Laplacian matrix. DCGC proposed in this paper is built on top of the graph Laplacian $\mathbf{K} = \mathbf{L}$ due to its simplicity.

The graph diffusion process defined above offers a principled and intuitive way to define the distribution of graph data by modeling the evolution of node signals over time. Given an initial node signal $\mathbf{H}(0)$, the diffused signal $\mathbf{H}(t) = e^{-\mathbf{L}t}\mathbf{H}(0)$ at time $t$ captures how information propagates through the graph structure By varying the diffusion time $t$, this process naturally encodes multi-scale information: small $t$ values capture local, low-order structural patterns, while large $t$ values reflect global, high-order dependencies. This property makes the diffusion process particularly suitable for defining graph distributions, as it seamlessly integrates both local and global graph characteristics. Using the graph diffusion process as a distribution definition not only provides a flexible and interpretable framework but also enables effective distribution matching by comparing the diffused signals of different graphs.

Given the above formulation, we define the distribution of a graph from the diffusion process at time $t$ as follows:

**Definition 1** *Given a graph $\mathcal{G}$ with initial node signals $\mathbf{H}(0) = \mathbf{H}$, the adjacency matrix $\mathbf{A}$, we define the **node signal distribution** over $\mathcal{G}$ at time $t$, termed $\mathbb{G}(t)$, as follows:*

$$\mathbb{G}(t) \triangleq \mathbf{H}(t),\ \boldsymbol{g}(t) \sim \mathbb{G}(t) \tag{4}$$

*where $\boldsymbol{g}(t)$ is a sample from distribution $\mathbb{G}(t)$, and denotes the signal of a node at time $t$. Specifically, $\boldsymbol{g}_i(t) = \boldsymbol{h}_i(t)$ is the $i$-th row of $\mathbf{H}(t)$.*

Similarly, for the synthetic graph $\mathcal{S}$, we use $\mathbb{S}(t)$, and $\boldsymbol{s}_i(t) \sim \mathbb{S}(t)$ to denote the elements at time $t$ correspondingly.

Based on Definition 1, we propose the following diffusion matching objective, aiming at matching the distribution of two graphs at any time $t$ during the diffusion process:

$$\min_{\mathcal{S}=(\mathbf{X}',\mathbf{A}')} \mathcal{D}(\mathbb{G}(t), \mathbb{S}(t)), \forall t > 0 \tag{5}$$

#### 3.2.2. Discretization of the Graph Diffusion Process

The graph diffusion process, defined by the continuous-time evolution of node signals $H(t)$, provides a powerful framework for capturing multi-scale structural information in graphs. However, since the diffusion time $t$ is continuous and unbounded, practical implementation requires discretization to obtain a finite set of time steps $\{t_0, t_1, \cdots, t_T\}$, where $T$ is the maximum number of steps. This discretization allows us to sample the diffusion process at specific intervals, balancing the trade-off between granularity and computational efficiency. By selecting an appropriate maximum time step $T$, we can ensure that the diffusion process captures both local and global structural properties without unnecessary computational overhead.

#### 3.2.3. Efficient computation via Euler's method

Computing the matrix exponential $\exp(-\mathbf{L}t)$ for large graphs can be computationally expensive, particularly when dealing with high-dimensional node features or large-scale graphs. To address this challenge, we propose using Euler's method to approximate the diffusion process in a discrete and efficient manner. Euler's method discretizes the continuous-time differential equation $\frac{\mathrm{d}H(t)}{\mathrm{d}t} = -\mathbf{L}\mathbf{H}(t)$ into the following iterative update rule

$$\mathbf{H}(t + \Delta t) = \mathbf{H}(t) - \Delta t \cdot \mathbf{L}\mathbf{H}(t) \tag{6}$$

where $\Delta t$ is the step size controlling the granularity of the discretization. This approach avoids the need for explicit computation of the matrix exponential, significantly reducing computational complexity. Moreover, Euler's method provides a flexible and scalable framework for simulating the diffusion process, making it suitable for large-scale graph analysis tasks.

The discretized diffusion process offers several advantages. First, it allows us to explicitly control the trade-off between computational efficiency and the fidelity of the diffusion approximation by adjusting the step size $\Delta t$ and the maximum time step $T$. Second, the iterative nature of Euler's method enables efficient computation $\mathbf{H}(t)$ at multiple time steps, facilitating the extraction of multi-scale graph representations. Finally, this approach is highly compatible with modern graph neural networks (GNNs), which often rely on discrete message-passing mechanisms that can be interpreted as approximations of the diffusion process. By leveraging these properties, we can effectively model the distribution of graph-structured data while maintaining computational traceability.

#### 3.2.4. Leveraging supervision

In supervised scenarios, each node is associated with a label, and we can leverage this additional information to enhance the diffusion process. Let $Y$ denote the label matrix, where each row corresponds to the one-hot label (or label distribution) of a node. Similar to node features, $Y$ can also be treated as a node signal that diffuses over the graph structure. This approach is conceptually aligned with label propagation algorithms (Xiaojin and Zoubin, 2002), which propagate label information through the graph to infer labels for unlabeled nodes.

To ensure synchronization between the diffusion of node features $\mathbf{X}$ and labels $\mathbf{Y}$, we propose concatenating $\mathbf{X}$ and $\mathbf{Y}$ into a unified node signal matrix $\mathbf{H} = [\mathbf{X} \| \mathbf{Y}] \in \mathbb{R}^{N \times (D+C)}$.

This combined matrix $\mathbf{H}$ is then used as the input to the graph diffusion process. By diffusing $\mathbf{H}$, we simultaneously propagate both node features and label information, allowing the model to capture the interplay between attribute and label distributions. This joint diffusion process not only preserves the structural relationships between nodes but also enhances the consistency between feature and label representations.

In addition to concatenating the node labels, in the supervised setting, we also consider the class-wise diffusion matching objective, which has the following formulation for time $t$:

$$\min_{\mathcal{S}=(\mathbf{X}',\mathbf{A}')} \mathcal{D}(\mathbb{G}_c(t), \mathbb{S}_c(t)), \ , c = 1, 2, \cdots, C, \tag{7}$$

where $\mathcal{G}_c(t)$ and $\mathcal{S}_c(t)$ are the class-wise original/synthetic graph distribution. To be specific, $\mathcal{G}_c(t) \triangleq \mathbf{H}_c(t)$ contains only nodes belonging to class $c$. The class-wise diffusion matching loss focuses on matching the node signal distribution of each individual class, thereby learning more discriminative synthetic graphs that are beneficial for node classification tasks.

## 3.3. Training of DCGC

### 3.3.1. Initialization

Given a condensation ratio $r$, the number of nodes in the condensed graph is $N' = N \times r$. Then, in the supervised setting, we initialize the labels of a condensed graph $\mathbf{Y}'$ such that the proportion of each class in the condensed graph is the same as that in the original full graph, i.e., $\frac{N'_c}{N'} = [\frac{N_c}{N}]$, and $\sum_c N'_c = N$. Note that in the unsupervised setting, we do not have to initialize the labels. The initialization of node features $\mathbf{X}'$ and adjacency matrix $\mathbf{A}'$ of the synthetic graph $\mathcal{S}$ is important to the optimization process, Empirically, we found that the traditional random initialization methods (e.g., Xavier initialization) lead to slow convergence speed and poor performance. To this end, we adopt a simple strategy to initialize the node feature matrix $\mathbf{X}'$ and the graph adjacency matrix $\mathbf{A}'$. For each class $c$, we randomly select $N'_c$ nodes from the original graph having the same label and use their features to initialize $\mathbf{X}'_c$. In this way, we wish the synthetic graph had individual node features similar to those of the original graph.

For the adjacency matrix $\mathbf{A}'$, we use a learnable matrix $\mathbf{P} \in \mathbb{R}^{N' \times N'}$ to parameterize $\mathbf{A}'$ and initialize $\mathbf{P}$ such that the obtained $\mathbf{A}'$ exhibits desired properties. Given $\mathbf{P}$, we obtain $\mathbf{A}' = \sigma(\mathbf{P} + \mathbf{P}^\top)$ such that $\mathbf{A}'$ is a symmetric matrix, and any entry is in the range of $(0, 1)$. $\sigma(\cdot)$ is the sigmoid function. We initialize $\mathbf{P}$ such that the on-diagonal terms of $\mathbf{A}'$ to be a value $\epsilon_{\mathsf{on}}$ close to 1, while off-diagonal terms to be a small value $\epsilon_{\mathsf{off}}$ close to 0. In this way, we initialize a synthetic graph that primarily consists of self-loops, thereby reducing the noisy edges that random initialization may introduce.

### 3.3.2. Distribution matching with MMD loss

The proposed model DCGC seeks to learn the synthetic graph by minimizing the MMD between the distribution $\mathbb{G}$ of the original full graph $\mathcal{G}$, and the distribution $\mathbb{S}$ of the synthetic graph $\mathcal{S}$ given a class $c$ (if node labels are available) and time $t$:

$$\begin{aligned}\mathcal{L}_c(t) &= \mathsf{MMD}^2(\mathbb{G}_c(t), \mathbb{S}_c(t)) \\ &= \mathop{\mathbb{E}}_{\mathbb{G}} \kappa(\boldsymbol{g}_c(t), \boldsymbol{g}'_c(t)) + \mathop{\mathbb{E}}_{\mathbb{S}} \kappa(\boldsymbol{s}_c(t), \boldsymbol{s}'_c(t)) - 2 \mathop{\mathbb{E}}_{\mathbb{G},\mathbb{S}} \kappa(\boldsymbol{g}_c(t), \boldsymbol{s}_c(t)) \\ &\Rightarrow \sum_{i=1}^{N'_c}\sum_{j=1}^{N'_c} \kappa(\boldsymbol{s}_{c,i}(t), \boldsymbol{s}_{c,j}(t)) - 2 \sum_{i=1}^{N_c}\sum_{j=1}^{N'_c} \kappa(\boldsymbol{g}_{c,i}(t), \boldsymbol{s}_{c,j}(t)).\end{aligned} \tag{8}$$

The last step discards the term $\mathop{\mathbb{E}}_{\mathbb{G}} \kappa(\boldsymbol{g}_c(t), \boldsymbol{g}'_c(t))$ since it only depends on the original graph $\mathcal{G}$ and is not involved in the optimization process. Note that Eq. 8 specifies the class id $c$ in the supervised setting, while in the unsupervised setting we can neglect the subscript $c$. For the kernel $\kappa(\cdot,\cdot)$, we utilize the most widely-used Gaussian kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|_2^2}{2\sigma^2})$, and $\sigma$ is the bandwidth hyperparameter.

#### 3.3.3. Regularization on the adjacency matrix $\mathbf{A}'$

Note that the original graph $\mathcal{G}$ is an undirected and unweighted graph with a symmetric adjacency matrix $\mathbf{A}'$, and each entry $A'_{ij} \in \{0, 1\}$. Therefore, we apply an additional regularization loss function directly on the learned adjacency matrix $\mathbf{A}'$, encouraging each entry to be close to either 0 or 1:

$$\mathcal{L}_{reg} = \sum_{i=1}^{N'}\sum_{j=1}^{N'} A'_{ij}(1 - A'_{ij}). \tag{9}$$

After the training process ends, we sparsify $\mathbf{A}'$ such that each entry is binarized to $\{0, 1\}$ according to whether $A'_{ij}$ is larger or smaller than 0.5.

#### 3.3.4. Overall objective function

The overall learning objective is the weighted summation of the diffusion matching loss and the regularization loss. Formally,

$$\min_{\mathbf{X}',\mathbf{A}'} \mathcal{L} = \sum_{t=0}^{T}\sum_{c=1}^{C} \mathcal{L}_c(t) + \lambda \cdot \mathcal{L}_{reg}, \tag{10}$$

where $\lambda$ is the trade-off hyperparameter.

#### 3.3.5. Complexity Analysis

Finally, we analyze the complexity of DCGC. We use $D_H$ to denote the dimension of $\mathbf{H}$, which will vary for the unsupervised and supervised settings. Also, we use $E$ and $E'$ to denote the number of edges in the original graph $\mathcal{G}$ and the synthetic graph $\mathcal{S}$, respectively. Note that there is $E \ll N^2$, and $E' <= N'^2$. The training cost comes from three parts: 1) Computing the signal distribution of the original graph $\mathcal{G}$ requires $\mathcal{O}(ETD_H)$, while this process is non-parametric and can be obtained via one-step preprocessing. Therefore, the computation overhead of this step is negligible compared with the entire condensation process. 2) Computing the signal distribution of the condensed graph requires $\mathcal{O}(E'TH) =$

$\mathcal{O}(N'^2TH)$. 3) Computing the MMD loss for all $t$ and $c$ takes $\mathcal{O}(T \cdot C \cdot \sum_{c=1}^{C} N'_c(N_c + N'_c))$, which depends on the number of nodes in each class. Yet, notice that $\sum_{c=1}^{C} N'_c(N_c + N'_c) \leq \sum_{c=1}^{C} N'_c(N + N') = N'(N + N')$, and the equality holds if and only if there is only one class. Therefore, it is reduced to $\mathcal{O}(TCN'(N + N'))$. Considering that $N' \ll N$, and both $C$ and $T$ are small constants in practice, the overall complexity is slightly greater than $\mathcal{O}(N)$ and much smaller than $\mathcal{O}(N^2)$, and therefore DCGC is time and memory-efficient.

## 4. Experiments

In this section, we conduct experiments to compare the proposed DCGC with SOTA graph condensation methods. The goal of experiments is to answer the following research questions:

- **RQ1**: How is the quality of graphs condensed by DCGC? Can the condensed graphs preserve important properties of the original graphs?
- **RQ2**: What is the utility of the condensed graphs in training Graph Neural Networks? Can the condensed graph improve the training efficiency of GNNs without hurting their performance significantly?
- **RQ3**: Does DCGC demonstrate better capacities/properties ability compared with existing gradient-matching methods, e.g., in terms of the cross-architecture generalization abilities and training efficiency?

### 4.1. Experimental Setups

**Datasets.** Following previous literature (Jin et al., 2022; Liu et al., 2022), we conduct experiments on six node classification datasets: Cora, Citeseer, Pubmed (Yang et al., 2016), Flickr, Reddit (Zeng et al., 2020), and Ogbn-arXiv (Hu et al., 2020). For a fair comparison, we use the public splits for all datasets.

**Settings.** We consider both the unsupervised setting and the supervised setting. In the unsupervised setting, the node labels are unknown, and in this setting, we evaluate whether the condensed graphs can preserve important properties of the original graphs. While in the supervised setting, we evaluate the performance of GNNs when trained on the synthetic graphs using the given labels.

**Competitors.** We compare our proposed method with four SOTA graph condensation methods: GCond (Jin et al., 2022), GCDM (Liu et al., 2022), SGDD (Yang et al., 2023), and SFGC (Zheng et al., 2023). Following (Jin et al., 2022), we also compare with three traditional selection-based methods: Herding (Welling, 2009), K-center (Sener and Savarese, 2018), and graph coarsening (Huang et al., 2021). The training performance using the original full graph is provided for reference as well.

**Implementation Details.** We implement the proposed method with Pytorch and DGL (Wang et al., 2019). In the training stage, we first initialize the node feature matrix $\mathbf{X}'$ and $\mathbf{A}'$ according to the proposed strategies. Then $\mathbf{X}'$ and $\mathbf{A}'$ are optimized using Eq. 10. In the evaluation stage, we train a 2-layer GCN model (Kipf and Welling, 2017) of hidden dimension 512 using the condensed graph and then report the accuracy on the testing nodes of

Table 1: Comparison with SOTA methods regarding testing accuracy (%). **Bold entries are the best results**. DCGC outperforms existing methods on almost all datasets and all condemnation ratios.

| Dataset | Ratio ($r$) | Other graph size reduction methods | | | Condensation Methods | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Herding | K-Center | Coarsening | GCond | GCDM | SGDD | SFGC | GDEM | GEOM | EXGC | GCSR | DCGC | Whole |
| **Cora** | 1.30% | 67.0±1.3 | 64.0±2.3 | 31.2±0.2 | 79.8±1.3 | 69.4±1.3 | 80.1±0.7 | 80.1±0.5 | 80.7±0.6 | 82.5±0.4 | 81.9±1.0 | 79.9±0.7 | **82.7±0.6** | |
| | 2.60% | 73.4±1.0 | 73.2±1.2 | 65.2±0.6 | 80.1±0.6 | 77.2±0.4 | 80.6±0.8 | 81.9±0.5 | 81.2±0.5 | **83.6±0.3** | 82.3±0.9 | 80.6±0.8 | 83.0±0.6 | 82.7±0.5 |
| | 5.20% | 76.8±0.1 | 76.7±0.1 | 70.6±0.1 | 79.3±0.3 | 79.4±0.1 | 80.4±1.6 | 81.6±0.8 | 81.3±0.5 | 82.8±0.7 | 82.6±0.5 | 81.2±0.6 | **83.1±0.5** | |
| **Citeseer** | 0.90% | 57.1±1.5 | 52.4±2.8 | 52.2±0.4 | 70.5±1.2 | 62.0±0.1 | 69.5±0.4 | 71.4±0.5 | 72.3±0.3 | **73.0±0.5** | 69.7±1.5 | 70.2±0.6 | 72.6±0.6 | |
| | 1.80% | 66.7±1.0 | 64.3±1.0 | 59.0±0.5 | 70.6±0.4 | 69.5±1.1 | 70.2±0.8 | 72.4±0.4 | 72.6±0.6 | **74.3±0.1** | 70.1±0.7 | 71.7±0.9 | 73.1±0.5 | 72.4±0.4 |
| | 3.60% | 69.0±0.1 | 69.1±0.1 | 65.3±0.5 | 69.8±1.4 | 69.8±0.2 | 70.3±1.7 | 70.6±0.7 | 72.6±0.5 | 73.3±0.4 | 70.5±0.9 | 74.0±0.4 | **74.2±0.5** | |
| **Pubmed** | 0.08% | 76.7±0.7 | 64.5±2.7 | 18.1±0.1 | 76.5±0.2 | 75.7±0.3 | 76.7±0.4 | 77.1±0.5 | 77.7±0.7 | 78.1±0.5 | 77.9±1.1 | 77.8±0.8 | **78.4±0.5** | |
| | 0.15% | 76.2±0.5 | 69.4±0.7 | 28.7±4.1 | 77.1±0.5 | 77.3±0.1 | 77.5±0.4 | 77.6±0.5 | 78.4±1.8 | 78.4±0.6 | 78.1±0.9 | 78.2±0.7 | **78.9±0.3** | 79.8±0.4 |
| | 0.30% | 78.0±0.5 | 69.1±0.1 | 65.3±0.5 | 77.9±1.4 | 78.3±0.9 | 78.2±0.8 | 78.8±0.6 | 78.2±0.8 | 78.7±0.6 | 78.2±0.8 | 78.4±0.6 | **79.5±0.3** | |
| **Flickr** | 0.10% | 42.5±1.8 | 42.0±0.7 | 41.9±0.2 | 46.5±0.4 | 46.8±0.2 | 46.9±0.1 | 46.6±0.6 | 46.9±0.8 | 47.1±0.1 | 47.0±0.1 | 46.6±0.3 | **47.6±0.3** | |
| | 0.50% | 43.9±0.9 | 43.2±0.1 | 44.5±0.1 | 47.1±0.1 | 47.9±0.3 | 47.1±0.3 | 47.0±0.1 | 47.1±1.3 | 47.0±0.2 | 48.3±0.5 | 46.6±0.2 | **48.2±0.3** | 50.2±0.3 |
| | 1.00% | 44.4±0.6 | 44.1±0.4 | 44.6±0.1 | 47.1±0.1 | 47.5±0.1 | 47.1±0.1 | 47.1±0.1 | 47.2±0.6 | 47.3±0.3 | 48.4±0.9 | 46.8±0.2 | **48.9±0.1** | |
| **Reddit** | 0.05% | 53.1±2.5 | 46.6±2.3 | 40.9±0.5 | 88.0±1.8 | 86.5±1.1 | 90.5±2.1 | 89.7±0.2 | 90.8±0.3 | **91.3±0.4** | 89.9±0.1 | 90.5±0.2 | 90.9±1.4 | |
| | 0.10% | 62.7±1.0 | 53.0±3.3 | 42.8±0.8 | 89.6±0.7 | 88.3±0.8 | 91.6±1.0 | 90.0±0.3 | 91.3±0.2 | 91.4±0.2 | 90.2±0.1 | 91.2±0.2 | **91.7±0.9** | 93.9±0.0 |
| | 0.20% | 71.0±1.6 | 58.5±2.1 | 47.4±0.9 | 90.1±0.5 | 89.2±0.7 | 91.6±1.8 | 89.9±0.4 | 91.7±0.4 | 91.5±0.4 | 90.6±0.9 | 92.2±0.1 | **92.5±0.6** | |
| **arXiv** | 0.05% | 52.4±1.8 | 47.2±3.0 | 35.4±0.3 | 59.2±1.1 | 56.2±0.3 | 60.8±1.3 | 65.5±0.7 | 63.7±0.8 | 65.5±0.6 | 57.6±0.6 | 60.6±1.1 | **66.8±0.7** | |
| | 0.25% | 58.6±1.2 | 56.8±0.8 | 43.5±0.2 | 63.2±0.3 | 59.6±0.4 | 65.8±1.2 | 66.1±0.4 | 63.8±0.6 | **68.8±0.2** | 62.3±0.3 | 65.4±0.8 | 68.1±0.6 | 71.4±0.1 |
| | 0.50% | 60.4±0.8 | 60.3±0.4 | 50.4±0.1 | 64.0±0.4 | 62.4±0.1 | 66.3±0.8 | 66.8±0.4 | 64.1±0.3 | **69.6±0.2** | 65.0±0.8 | 65.9±0.6 | 68.9±0.4 | |

the original graph. We repeat all experiments 20 times and report the average performance with standard deviation.

**Hyperparameter settings.** For the initialization of the adjacency matrix $\mathbf{A}'$, we set $\varepsilon_{\mathsf{on}} = 0.999.$ and $\varepsilon_{\mathsf{off}} = 0.001$. The diffusion time interval is set as $\Delta t = 1$, and the maximum diffusion step is set as $T = 5$. For the bandwidth of the Gaussian kernel function when computing the MMD distance, we set $2\sigma^2$ as the median $\ell_2$ distance of the samples since it is dataset-sensitive. $\lambda = 1e-3$ for all datasets.

### 4.2. Utility in Training GNNs

**Comparison on common benchmarks.** In Table 1, we present the performance comparison between the proposed DCGC and the baseline methods under node classification tasks. The experimental results demonstrate that our proposed method performs on par or even better than SOTA gradient-matching methods on all datasets and condensation ratios, which strongly illustrates the effectiveness of DCGC across different datasets.

**Cross-architecture generalization performance.** One important limitation of existing methods is that they all rely on a predefined GNN encoder during the condensation process, which might lead to poor cross-architecture generalization ability. In this section, we empirically validate the generalization ability of the proposed DCGC on Cora, Citeseer, Pubmed, and Ogbn-arXiv. The condensation process of DCGC involves no encoders. In evaluation, we consider different-architectured GNN classifiers: GCN (Kipf and Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Velickovic et al., 2018), and APPNP (Klicpera et al., 2019). We also report the average performance with standard deviation across different architectures. A small standard deviation indicates that the condensed graph has relatively stable performance across classifiers with different architectures, so a model with a higher average accuracy and a smaller standard deviation is preferred. As demonstrated in Table 2, the proposed DCGC achieves high average accuracy with low Std. across different GNN architectures. This indicates DCGC 's superior generalization ability

Table 2: Cross-architecture generalization performance comparison. The condensed graphs are obtained via GCN (except DCGC, which is data-centric), while tested using six different GNN architectures: GCN, SAGE, GAT, and APPNP, and the overall performance is reflected by the average testing accuracy (Avg.) and its standard deviation (Std.).

| **Datasets** | **Methods** | **Architectures** | | | | **Statistics** | |
|---|---|---|---|---|---|---|---|
| | | **GCN** | **SAGE** | **GAT** | **APPNP** | Avg. | Std. |
| **Cora** $r = 2.6\%$ | GCond | 80.1 | 78.2 | 66.2 | 78.5 | 75.8 | 6.42 |
| | GCDM | 79.4 | 78.5 | 73.2 | 77.8 | 77.2 | 2.76 |
| | SGDD | 79.8 | 80.4 | 75.8 | 78.4 | 78.6 | 2.05 |
| | SFGC | 81.1 | 81.9 | 80.8 | 78.8 | 80.6 | 1.31 |
| | GDEM | 81.2 | 80.3 | 80.5 | 82.1 | 81.0 | 0.81 |
| | GEOM | 83.6 | 83.7 | 82.7 | 81.9 | 83.0 | 0.84 |
| | DCGC | 83.0 | 83.2 | 82.7 | 83.3 | **83.1** | **0.27** |
| **Citeseer** $r = 1.8\%$ | GCond | 70.6 | 66.2 | 55.4 | 69.6 | 65.5 | 6.96 |
| | GCDM | 69.5 | 67.1 | 62.5 | 69.1 | 67.1 | 3.21 |
| | SGDD | 70.2 | 67.8 | 65.7 | 70.7 | 68.6 | 2.31 |
| | SFGC | 71.6 | 71.7 | 72.1 | 70.5 | 71.5 | 0.68 |
| | GDEM | 72.6 | 71.7 | 71.5 | 72.1 | 72.0 | 0.49 |
| | GEOM | 74.3 | 74.1 | 74.2 | 74.0 | **74.2** | **0.13** |
| | DCGC | 73.2 | 72.5 | 72.9 | 72.7 | 72.8 | 0.30 |
| **Pubmed** $r = 0.15\%$ | GCond | 77.1 | 76.2 | 74.8 | 77.9 | 76.5 | 1.33 |
| | GCDM | 77.3 | 75.7 | 77.9 | 78.2 | 77.3 | 1.11 |
| | SGDD | 77.5 | 76.9 | 76.8 | 78.7 | 77.5 | 0.87 |
| | SFGC | 77.6 | 77.4 | 77.1 | 78.2 | 77.6 | 0.46 |
| | GDEM | 78.4 | 77.1 | 76.9 | 78.1 | 77.6 | 0.74 |
| | GEOM | 78.7 | 77.2 | 77.5 | 78.9 | 78.1 | 0.85 |
| | DCGC | 78.9 | 78.6 | 79.4 | 79.5 | **79.1** | **0.42** |

across different architectures. These results clearly demonstrate the superior advantages of DCGC as a data-centric condensation method.

### 4.3. Ablation Studies and Efficiency Comparison

Table 3: Performance of removing feature signal /label signal /regularization loss on Ogbn-arXiv dataset.

| Variants | $r = 0.05\%$ | $r = 0.25\%$ | $r = 0.50\%$ |
|---|---|---|---|
| w/o label signal | 56.1 | 59.8 | 61.1 |
| w/o feature signal | 34.3 | 39.9 | 43.2 |
| w/o $\mathcal{L}_{reg}$ | **66.9** | **68.5** | **69.2** |
| DCGC | 66.8 | 68.1 | 68.9 |

**Effects of the components in DCGC.** Next, we investigate the importance of each component of DCGC. The loss function of DCGC (in Eq. 10) consists of three parts: feature-level signal, label-level signal, and regularization loss, while the last only takes effect when both the former ones exist. Therefore, we investigate the impact of using each individual loss

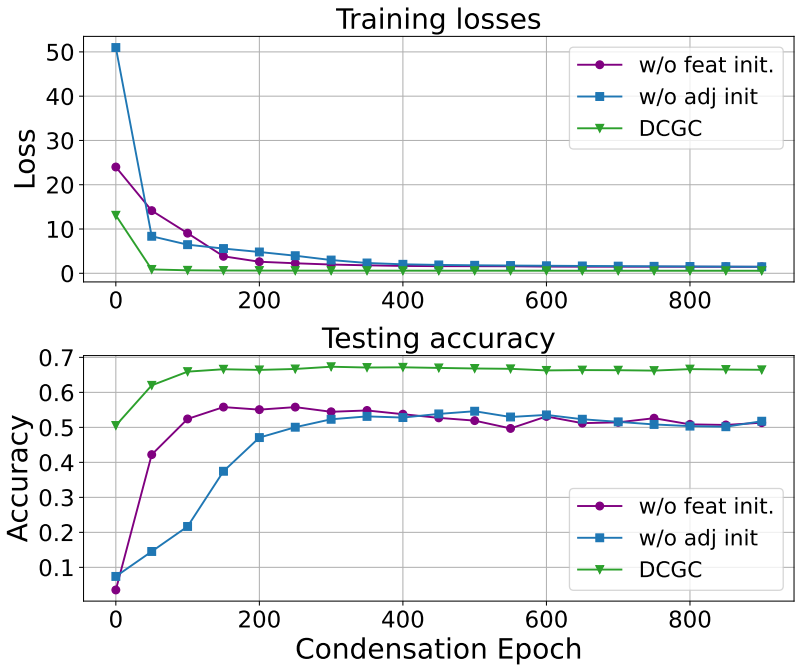


Figure 3: Ablation studies on initialization strategies

separately on the performance of DCGC. In Table 3, we present the results on Ogbn-arXiv dataset. It is observed that using merely the feature signal or label signal can lead to sub-optimal performance. This indicates that solely considering the feature distribution or label distribution over the graph is insufficient to capture the distribution of the entire graph, especially when the graph's structure is complex. In addition, an interesting observation is that adding the regularization loss $\mathcal{L}_{reg}$ on the synthetic graph's adjacency matrix $\mathbf{A}'$ slightly impair its utility in training GNNs. However, this step is necessary for obtaining a reasonable sparse graph structure.

**Effects of the DCGC's initialization strategies.** Next, we investigate the importance of the initialization strategies, which are assessed by removing the feature matrix initialization and adjacency matrix initialization from DCGC, respectively. In Fig. 3, we present the training curves of training loss and test accuracy w.r.t. the epoch on Ogbn-arXiv dataset ($r = 0.5\%$). It can be observed that with the proposed two initialization strategies, the initial loss is set to be very low, resulting in a good starting point in the optimization space. This not only significantly accelerates the model's convergence speed but also makes it easier for the model to converge to better values, reducing the risk of getting stuck in local optima. Removing any one of the initialization methods significantly increases the training difficulty of the model, which may lead to sub-optimal performance.

**Comparison of training time.** Finally, we validate the efficiency of the proposed DCGC by comparing its training time with SOTA graph condensation methods. Following previous evaluation settings (Jin et al., 2022; Yang et al., 2023), we report the training time of 50 epochs on Ogbn-arXiv dataset in Table 4. As shown in Table 4, DCGC achieves a much faster training speed compared with existing methods for all condensation ratios. To

Table 4: Training time comparison on Ogbn-arXiv dataset.

| $r$ | GCond | GCDM | SGDD | GDEM | DEOM | DCGC |
|---|---|---|---|---|---|---|
| 0.05% | 351 s | 325 s | 349 s | 47 s | 437 s | **11.69 s** |
| 0.25% | 448 s | 358 s | 417 s | 59 s | 482 s | **12.21 s** |
| 0.50% | 603 s | 411 s | 576 s | 64 s | 695 s | **13.84 s** |

be specific, DCGC reduces the epoch-wise training time by 96.4% compared with SOTA gradient matching methods (note that GDEM is also distribution-matching based). Furthermore, as the graph condensation $r$ increases, the training time of DCGC increases to a lesser extent compared to other methods. This indicates that our proposed DCGC exhibits better scalability relative to other methods.

## References


Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *International Conference on Learning Representations*, 2021.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

Junfeng Fang, Xinglin Li, Yongduo Sui, Yuan Gao, Guibin Zhang, Kun Wang, Xiang Wang, and Xiangnan He. Exgc: Bridging efficiency and explainability in graph condensation. In *Proceedings of the ACM on Web Conference 2024*, pages 721–732, 2024.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Mridul Gupta, Sahil Manchanda, HARIPRASAD KODAMANA, and Sayan Ranu. Mirage: Model-agnostic graph distillation for graph classification. In *The Twelfth International Conference on Learning Representations*, 2024.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 675–684, 2021.

Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. In *International Conference on Learning Representations*, 2022.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

Risi Kondor and Jean-Philippe Vert. Diffusion kernels. *kernel methods in computational biology*, pages 171–192, 2004.

Manoj Kumar, Anurag Sharma, Shashwat Saxena, and Sandeep Kumar. Featured graph coarsening with similarity guarantees. In *International Conference on Machine Learning*, pages 17953–17975. PMLR, 2023.

Mengyang Liu, Shanchuan Li, Xinshi Chen, and Le Song. Graph condensation via receptive field distribution matching. *arXiv preprint arXiv:2206.13697*, 2022.

Zhanyu Liu, Chaolv Zeng, and Guanjie Zheng. Graph data condensation via self-expressive graph structure reconstruction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1992–2002, 2024.

Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *International Conference on Machine Learning*, pages 3237–3246. PMLR, 2018.

Manon Réau, Nicolas Renaud, Li C Xue, and Alexandre MJJ Bonvin. Deeprank-gnn: a graph neural network framework to learn patterns in protein–protein interfaces. *Bioinformatics*, 39(1):btac759, 2023.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

Hannes Stärk, Dominique Beaini, Gabriele Corso, Prudencio Tossou, Christian Dallago, Stephan Günnemann, and Pietro Liò. 3d infomax improves gnns for molecular property prediction. In *International Conference on Machine Learning*, pages 20479–20502. PMLR, 2022.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv*, 1909.01315, 2019.

Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the diffusion process in linear graph convolutional networks. *Advances in Neural Information Processing Systems*, 34:5758–5769, 2021.

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

Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Peng He, Paul Weng, Han Gao, and Guihai Chen. Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems. In *WWW*, pages 2091–2102, 2019.

Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. Difformer: Scalable (graph) transformers induced by energy constrained diffusion. In *The Eleventh International Conference on Learning Representations*, 2023.

Zhu Xiaojin and Ghahramani Zoubin. Learning from labeled and unlabeled data with label propagation. 2002.

Beining Yang, Kai Wang, Qingyun Sun, Cheng Ji, Xingcheng Fu, Hao Tang, Yang You, and Jianxin Li. Does graph distillation see like vision dataset counterpart? In *NeurIPS*, 2023.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2020.

Yuchen Zhang, Tianle Zhang, Kai Wang, Ziyao Guo, Yuxuan Liang, Xavier Bresson, Wei Jin, and Yang You. Navigating complexity: Toward lossless graph condensation via expanding window matching. *arXiv preprint arXiv:2402.05011*, 2024.

Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6514–6523, 2023.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021.

Xin Zheng, Miao Zhang, Chunyang Chen, Quoc Viet Hung Nguyen, Xingquan Zhu, and Shirui Pan. Structure-free graph condensation: From large-scale graphs to condensed graph-free data. *arXiv preprint arXiv:2306.02664*, 2023.