Data-Centric Graph Condensation via Diffusion Matching

Hengrui Zhang HZHAN55@UIC.EDU and **Philip S. Yu** PSYU@UIC.EDU
Chicago, IL, United States

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 provide a ubiquitous representation for systems of interactions, from 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). The recent success of deep learning methods on graph-structured data, particularly Graph Neural Networks (GNNs) (Kipf and Welling, 2017; Velickovic et al., 2018), has garnered significant attention. However, training deep GNNs on large-scale, real-world graphs requires tremendous computational and infrastructural resources due to the necessity of performing layer-by-layer message passing among interconnected nodes (Zeng et al., 2020).

To address this challenge, a natural approach is to compress the graph data. Traditional methods include graph sparsification (Spielman and Teng, 2011) and graph coarsening (Loukas and Vandergheynst, 2018b; Cai et al., 2021; Kumar et al., 2023), but these often rely on predefined heuristics and lack guidance from the training process (Yang et al., 2023), leading to suboptimal performance on downstream tasks. A more recent and successful technical path is 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), a synthesis-based approach that directly learns a small, dense synthetic graph. These methods are typically guided by gradient-matching (Zhao et al., 2021), aiming to learn a synthetic graph that replicates the gradient trajectory of model parameters observed when training on the original graph.

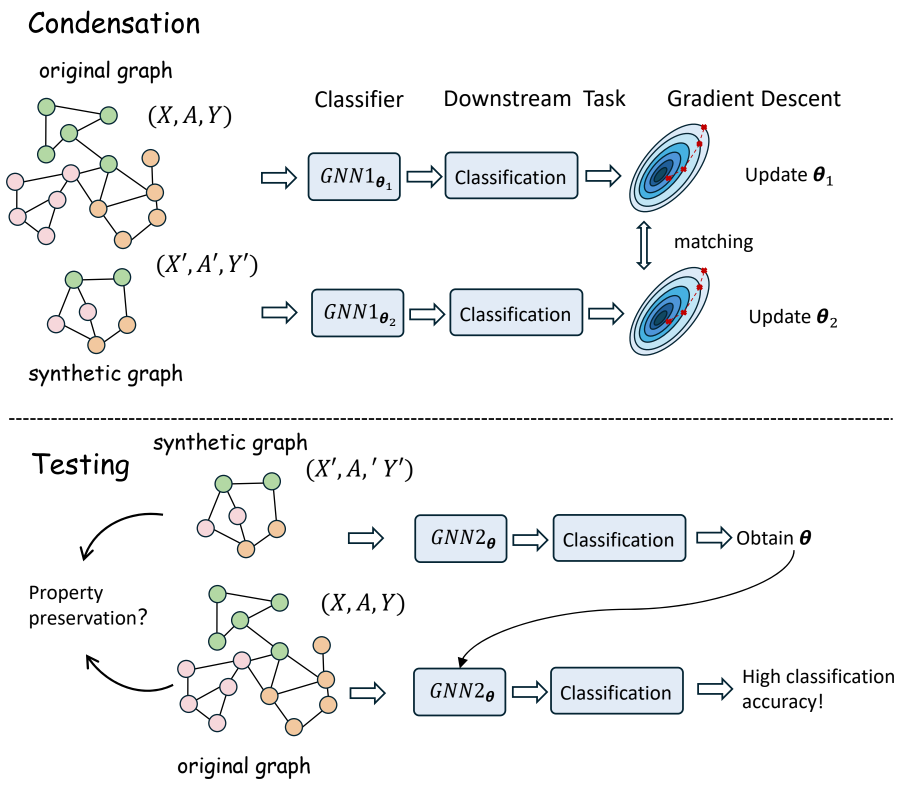

Figure 1: Limitations of existing gradient matching methods. Top: Gradient matching is supervised, task-oriented and time-consuming. The success of gradient matching relies on an accurate matching of the gradient update trajectories. Bottom: In testing, another GNN has to be trained on the synthetic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

Although these gradient-matching methods have achieved promising performance, their design philosophy leads to unsatisfactory capabilities (Gupta et al., 2024) for several reasons:

- **Task- and Model-Oriented Nature.** Gradient matching is inherently coupled to a specific downstream task (e.g., node classification) and a predefined GNN architecture. This dependency limits its applicability in unsupervised or semi-supervised scenarios and means the resulting synthetic graph may not generalize well to other tasks or models.
- **Lack of Graph Property Preservation.** While the method ensures similar task performance, it does not guarantee that the synthetic graph preserves the intrinsic structural and attribute properties of the original graph, such as community structure or node degree distribution.

- **Computational Inefficiency.** The primary motivation for graph condensation is to improve efficiency. However, gradient-matching methods involve a complex and expensive bi-level optimization process that requires repeatedly training a GNN on the original large graph, which undermines the very goal of efficiency.

To address these limitations, this paper proposes Data-Centric Graph Condensation via Diffusion Matching (DCGC). Our approach moves away from gradient matching and instead embraces the principle of distribution matching (Zhao and Bilen, 2023), learning the condensed graph by minimizing the divergence between the distributions of the original and synthetic graphs. Specifically, we draw an analogy between a geometric diffusion process and message passing in GNNs. This allows us to decompose a graph into a collection of node signals, where each signal is an aggregation of its multi-order structural information. The distribution of a graph is then defined by the distribution of these aggregated signals, and the divergence is efficiently measured and optimized using the Maximum Mean Discrepancy.

DCGC resolves the limitations of prior work in the following ways. First, our approach is task-agnostic and model-agnostic, as it relies solely on the intrinsic properties of the graph captured by the diffusion process. Second, by matching the diffusion trajectories of node features and labels (if available), our method ensures the synthetic graph preserves both local and global structural properties. Finally, our approach avoids the inefficient bi-level optimization of gradient matching, and because the diffusion process is independent of any specific GNN, the resulting synthetic graphs exhibit strong cross-architecture generalization.

We evaluate DCGC on eight graph datasets of varying scales. Experimental results demonstrate that our condensed graphs not only preserve important graph properties but also yield comparable or even better performance than state-of-the-art gradient-matching methods. Crucially, in cross-architecture settings and on heterophilic datasets, DCGC exhibits superior and more stable performance, reducing the cross-architecture standard deviation by an average of 26.3%. In terms of efficiency, DCGC reduces the training time by 96.4% compared to the fastest existing methods. These results clearly demonstrate the superiority of DCGC in terms of efficacy, generalization, 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)

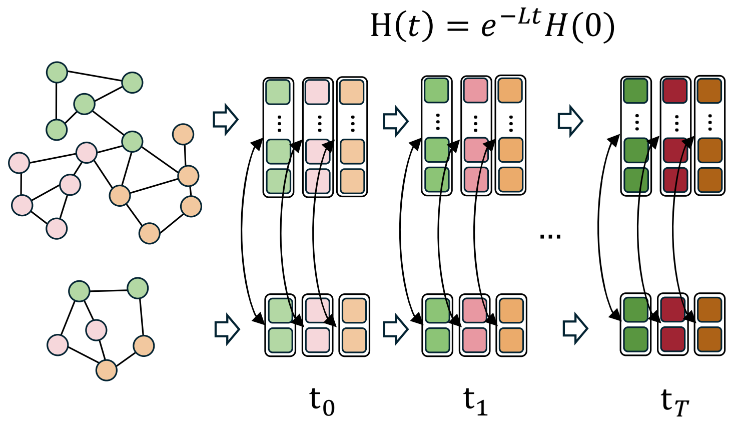

Figure 2: Illustration of graph diffusion process and diffusion matching. The distribution of a graph is defined as the diffusion trajectories of the node signal (e.g., node features/labels).

(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:

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

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 σ is the bandwidth parameter.

3. Methodology

3.1. Condensation via Distribution Matching

A natural approach to property-preserving graph condensation is to ensure the distribution of the synthetic graph ($\mathbb{S}$) closely resembles that of the original graph ($\mathbb{G}$) (Zhao and Bilen, 2023). This goal can be formulated as minimizing a divergence measure $\mathcal{D}$ between the two distributions:

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

However, defining a meaningful distribution for a graph is inherently challenging. Unlike traditional data types (e.g., images) where samples are often independent and identically distributed (i.i.d.), graph data is non-Euclidean and exhibits strong dependencies. A node's features are influenced by its neighbors and the graph topology, making them non-i.i.d. This interdependence necessitates a joint modeling of attributes and structure, which is non-trivial. Consequently, traditional distribution matching techniques that rely on i.i.d. assumptions (Zhao and Bilen, 2023) are inadequate for graph-structured data. The core challenge is thus to find a proper way to characterize the distribution of a graph and to quantify the divergence between two such distributions.

3.1.1. Node Diffusion Trajectories as a Measurement of Graph Distribution

To resolve the challenge of defining a graph distribution, we turn to the graph diffusion process (Kondor and Vert, 2004; Wang et al., 2021; Wu et al., 2023). This process utilizes a diffusion ODE to characterize the evolution of a graph signal (e.g., node features) under the spatial constraints of the graph structure. Formally, let $\mathbf{H}$ be the node signal; its diffusion process is defined as:

$$\begin{aligned}\frac{\mathrm{d}\mathbf{H}(t)}{\mathrm{d}t} &= -\mathbf{K}\mathbf{H}(t),\ \mathbf{H}(0)=\mathbf{H}, && \text{(ODE)}\\ \mathbf{H}(t) &= \exp(-\mathbf{K}t)\,\mathbf{H}(0), && \text{(solution)}\end{aligned}$$

where $\mathbf{H}(t)$ is the node signal matrix at time t, and $\mathbf{K}$ is the generalized diffusion kernel. A representative kernel is the heat kernel (Kondor and Vert, 2004), $\mathbf{K}=\mathbf{L}$, where $\mathbf{L}$ is the (normalized) graph Laplacian. The DCGC method proposed in this paper is built upon $\mathbf{K}=\mathbf{L}$.

The graph diffusion process provides a principled way to define the graph distribution. Given an initial signal $\mathbf{H}(0)$, the diffused signal $\mathbf{H}(t)$ captures how information propagates through the structure. By varying t, this process naturally encodes multi-scale information: small t captures local patterns, while large t reflects global dependencies.

Based on this, we define the distribution of a graph at time t as follows:

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

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

where $\boldsymbol{g}(t)$ is a sample from $\mathbb{G}(t)$, denoting the signal of a node at time t (i.e., a row of $\mathbf{H}(t)$).

For the synthetic graph $\mathcal{S}$, we similarly use $\mathbb{S}(t)$. Our diffusion matching objective aims to match the distributions of the two graphs at all times t:

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

3.1.2. Discretization and Efficient Computation of Graph Diffusion

While the diffusion process provides a powerful framework, the continuous and unbounded time t must be discretized into a finite set of time steps $\{t_0, t_1, \cdots, t_T\}$ for practical implementation.

Furthermore, directly computing the matrix exponential $\exp(-\mathbf{L}t)$ is computationally prohibitive for large graphs. To address this, we use Euler's method for an efficient approximation. Euler's method discretizes the continuous differential equation 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 Δt is the step size. This approach avoids the matrix exponential, significantly reducing complexity, and allows us to balance efficiency and fidelity by adjusting Δt and T.

3.1.3. Leveraging Supervision

In supervised scenarios, nodes are associated with labels $\mathbf{Y}$ (e.g., one-hot vectors). We can treat these labels as another node signal that diffuses over the graph, which is conceptually aligned with label propagation algorithms (Xiaojin and Zoubin, 2002).

To synchronize the diffusion of features and labels, we concatenate them into a unified signal matrix $\mathbf{H} = [\mathbf{X} \| \mathbf{Y}] \in \mathbb{R}^{N \times (D+C)}$, which is then used as the initial input $\mathbf{H}(0)$ to the diffusion process.

Additionally, in the supervised setting, we introduce a **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 $\mathcal{G}_c(t) \triangleq \mathbf{H}_c(t)$, contains only the node signals belonging to class c. This class-wise loss focuses on matching the signal distribution for each class individually, thereby learning a more discriminative synthetic graph beneficial for node classification tasks.

3.2. Training of DCGC

3.2.1. 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 ϵ_{on} close to 1, while off-diagonal terms to be a small value ϵ_{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.

3.2.2. 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 σ is the bandwidth hyperparameter.

3.2.3. 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.

3.2.4. 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 λ is the trade-off hyperparameter.

3.2.5. 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.

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), GDEM (Liu et al., 2023), GEOM (Zhang et al., 2024), and GCSR (Liu et al., 2024). 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

Table 1: Property preservation comparison in the unsupervised setting. The condensation ratios r for Cora, Citeseer, and Pubmed are 2.6%, 1.8% and 0.6%, respectively.

Metric	Dataset	GCDM	FGC	GDEM	DCGC
REE	**Cora**	0.092	0.078	0.073	**0.065**
	Citeseer	0.089	0.074	0.069	**0.067**
	Pubmed	0.152	0.139	0.128	**0.109**
DE in 10^4	**Cora**	0.197	0.214	0.146	**0.121**
	Citeseer	0.148	0.125	0.099	**0.092**
	Pubmed	0.175	0.149	0.138	**0.104**

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 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 ℓ_2 distance of the samples since it is dataset-sensitive. $\lambda = 1e - 3$ for all datasets.

4.2. Graph Property Preservation

In this section, we evaluate if the condensed graph can preserve the properties of the original graph well. Since this task is not relevant to node labels, we consider the unsupervised setting, and compare the performance of DCGC with other methods that do not require not labels. Following (Kumar et al., 2023), we consider the following metrics:

- **Relative Eigen Error (REE)** (Loukas and Vandergheynst, 2018a) measures the spectral similarity between two graphs. $REE = \frac{1}{m}\sum_{i=1}^{m}\frac{|\lambda_i - \lambda'_i|}{\lambda_i}$, where λ and λ'_i are the top m eigenvalues corresponding to the original graph Laplacian matrix $\mathbf{L}$ and condensed graph Laplacian matrix $\mathbf{L}'$, respectively. m is the eigenvalue number, and we set $m = 100$.

- The **Dirichlet energy (DE)** of the synthetic graph $\mathcal{S} = (\mathbf{X}', \mathbf{A}')$, which is defined according to $\mathcal{S}$'s feature matrix $\mathbf{X}'$ and Laplacian matrix $\mathbf{L}'$: $DE(\mathbf{X}', \mathbf{L}') = tr(\mathbf{X}'^{\top}\mathbf{L}'\mathbf{X}') = -\sum \mathbf{L}'_{ij}\|\boldsymbol{x}'_i - \boldsymbol{x}'_j\|^2$

For both metrics. lower values indicate that the condensed graphs have better preserved the properties of the original graph. The results are presented in Table 1, which demonstrates that the synthetic graph learned by DCGC can better preserve the properties of the original graph in terms of REE and DE.

Table 2: 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**	**GDEM**	**GEOM**	**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	80.7±0.6	81.9±1.0	79.9±0.7	**82.7±0.6**	82.7±0.5
	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	81.2±0.5	82.3±0.9	80.6±0.8	**83.0±0.6**	
	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	81.3±0.5	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	72.3±0.3	69.7±1.5	70.2±0.6	72.6±0.6	72.4±0.4
	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	72.6±0.6	70.1±0.7	71.7±0.9	**73.1±0.5**	
	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	72.6±0.5	70.5±0.9	**74.0±0.4**	73.0±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	77.7±0.7	77.9±1.1	77.8±0.8	**78.4±0.5**	79.8±0.4
	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	78.4±1.8	78.1±0.9	78.2±0.7	**78.9±0.3**	
	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	78.2±0.8	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	46.9±0.8	47.0±0.1	46.6±0.3	**47.6±0.3**	50.2±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	47.1±1.3	48.3±0.5	46.6±0.2	**48.2±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	47.2±0.6	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	90.8±0.3	89.9±0.1	90.5±0.2	**90.9±1.4**	93.9±0.0
	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	91.3±0.2	90.2±0.1	91.2±0.2	**91.7±0.9**	
	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	91.7±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	63.7±0.8	57.6±0.6	60.6±1.1	**65.2±0.7**	71.4±0.1
	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	63.8±0.6	62.3±0.3	65.4±0.8	**66.1±0.6**	
	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	64.1±0.3	65.0±0.8	65.9±0.6	**66.9±0.4**	

4.3. Utility in Training GNNs

Comparison on common benchmarks. In Table 2, 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 3, the proposed DCGC achieves high average accuracy with low Std. across different GNN architectures. This indicates DCGC 's superior generalization ability across different architectures. These results clearly demonstrate the superior advantages of DCGC as a data-centric condensation method.

Performance on heterophilic graphs. Another potential limitation of existing gradient-matching methods is that when the model used for graph condensation is suboptimal, the subsequent GNN model might suffer from significant performance degradation even if it has a proper architecture. To verify this, we perform experiments on two heterophilic datasets, Amazon-Rating and Actor (Pei et al., 2020). In Table 4, we present the

Table 3: 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
	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
	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
	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**

performance of GCond and SFGC using distinct GNN architectures for condensing heterophilic graphs. Note that GCN (Kipf and Welling, 2017) usually performs sub-optimally on heterophilic graphs, while GPR-GNN (Chien et al., 2021) is good at both homophilic and heterophilic graphs. We can observe that when using the same architecture for condensation and testing, all methods achieve close performance to training the architecture on the full graph. However, graphs condensed by GCN fail to give satisfying performance when evaluated using GPRGNN and are far from training GPRGNN on the full graph. By contrast, the proposed DCGC is able to give a consistently close performance to the full graph regardless of the architecture.

4.4. Ablation Studies and Efficiency Comparison

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 separately on the performance of DCGC. In Table 5, 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

Table 4: Performance of graph condensation methods on heterophilic graphs. The column **C** denotes the GNN model for condensation, while the row **T** denotes the GNN model for evaluation (training/testing). GNNs good at heterophilic graphs fail on graphs condensed by homophilic GNNs.

Datasets	**Methods**	Rating (r = 0.4%)		Actor (r = 1.3%)	
T/C		GCN	GPRGNN	GCN	GPRGNN
GCN	GCond	44.37	40.92	27.35	28.41
	SGDD	45.81	42.19	28.16	28.44
	GDEM	46.75		29.85	
	GEOM	45.92	42.10	29.06	29.26
	DCGC	**46.98**		**29.93**	
	Whole	48.70		30.59	
GPRGNN	GCond	41.18	41.59	31.26	36.68
	SGDD	41.92	42.58	33.39	37.74
	GDEM	43.52		38.52	
	GEOM	42.15	43.08	34.82	38.51
	DCGC	**43.71**		**38.77**	
	Whole	44.88		39.30	

Table 5: 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 $\mathcal{L}_{reg}$	**65.9**	**66.5**	**67.2**
DCGC	65.2	66.1	66.9

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 impairs 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

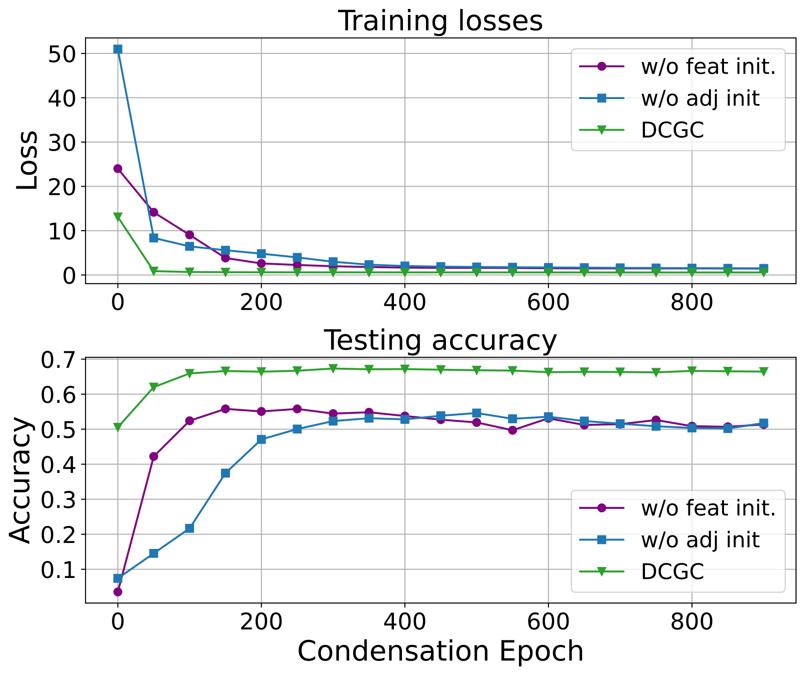

Figure 3: Ablation studies on initialization strategies

Table 6: 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**

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 6. As shown in Table 6, DCGC achieves a much faster training speed compared with existing methods for all condensation ratios. To 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.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 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.

Yang Liu, Deyu Bo, and Chuan Shi. Graph distillation with eigenbasis matching. *arXiv preprint arXiv:2310.09202*, 2023.

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, 2018a.

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

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

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.