A Preliminary

A.1 Graph Condensation Algorithms

For a graph learning problem, a graph is denoted as $\mathcal{G} = \{A, X, Y\}$, where the adjacency matrix $A \in \mathbb{R}^{n \times n}$ denotes the graph structure, and $X \in \mathbb{R}^{n \times d}$ is the d-dimensional feature matrix for n nodes. $Y \in \mathbb{R}^n$ represents node labels from a class set $\mathcal{C}$.

Graph condensation aims to synthesise a smaller graph $\mathcal{G}' = \{A', X', Y'\}$ from a larger original graph $\mathcal{G} = \{A, X, Y\}$ while ensuring that the model trained with $\mathcal{G}'$ performs similarly to the model trained with $\mathcal{G}$. This objective can be described as a bi-level optimisation problem:

$$\min_{\mathcal{G}'} \mathcal{L}_{\text{task}}(\mathcal{G}; \theta') \quad \text{s.t. } \theta' = \arg\min_{\theta} \mathcal{L}_{\text{task}}(\mathcal{G}'; \theta), \tag{2}$$

where the lower level optimisation aims to obtain the graph model weight, θ', by minimising the task-related loss, $\mathcal{L}_{\text{task}}$, on the condensed graph, $\mathcal{G}'$. With θ', the ultimate goal is to obtain the optimal $\mathcal{G}'$ that minimises $\mathcal{L}_{\text{task}}(\mathcal{G}; \theta')$ at the upper level. However, a direct tackling of this optimisation problem is nontrivial, which leads to a more convenient and effective matching-based paradigm:

$$\min_{\mathcal{G}'} \mathcal{L}_{\text{match}}(\mathcal{G}, \mathcal{G}'; \theta), \tag{3}$$

where $\mathcal{L}_{\text{match}}$ is the loss function to measure the distance of key statistics between $\mathcal{G}$ and $\mathcal{G}'$ given model parameter θ. Different GC methods can be categorised by the choice of statistics:

Gradient matching [13, 14, 30] methods match model learning gradients between the original graph $\nabla_\theta \mathcal{L}_{\text{task}}(\mathcal{G}; \theta)$ and the condensed graph $\nabla_\theta \mathcal{L}_{\text{task}}(\mathcal{G}'; \theta)$. The loss $\mathcal{L}_{\text{match}}$ can be defined as:

$$\mathcal{L}_{\text{match}} = \text{D}(\nabla_\theta \mathcal{L}_{\text{task}}(A, X, Y; \theta), \nabla_\theta \mathcal{L}_{\text{task}}(A', X', Y'; \theta)), \tag{4}$$

where $\text{D}(\cdot, \cdot)$ is a distance measure of gradients. Generally, the feature matrix X' is optimised directly with $\mathcal{L}_{\text{match}}$. To obtain the synthetic adjacency matrix A', GCond [14] generates A' by mapping X' with learnable MLPs,

$$A'_{i,j} = \text{Sigmoid}\left([\text{MLP}(x'_i \oplus x'_j) + \text{MLP}(x'_j \oplus x'_i)]/2\right), \tag{5}$$

where $\oplus$ represents the concatenate operation. Alternatively, A' can also be treated as the identity matrix I in the condensation process. SGDD [30] generates A' using a learnable generator GEN_ϕ from random noise $\mathcal{Z}$ and optimises this generator by reducing the optimal transport distance [3] between the Laplacian Energy Distributions [24] of A and A'. A more recent work GDEM [18] matches the eigenbasis between the condensed graph and the original graph to manage the structure learning.

Distribution matching [17, 19, 20] methods avoid the slow gradient calculation by aligning $\mathcal{G}$ and $\mathcal{G}'$ in the embedding space, corresponding to E for the original graph and E' for the condensed graph:

$$\mathcal{L}_{\text{match}} = \text{D}(E, E'), \tag{6}$$

where D can be implemented by distance measurement of empirical distribution of embeddings, such as Maximum Mean Discrepancy (MMD) in [19], $\sum_{c \in \mathcal{C}} \left\|\text{Mean}\left(E_c\right) - \text{Mean}\left(E'_c\right)\right\|^2$, where the subscript c denotes a specific class.

Trajectory matching [31, 32] methods aim to align the learning dynamics of models trained with $\mathcal{G}'$ and those with $\mathcal{G}$. This alignment is achieved by matching the parameters of models trained on $\mathcal{G}'$ with the optimal parameters of models trained on $\mathcal{G}$. The trajectory matching loss is defined as:

$$\mathcal{L}_{\text{match}} = \text{D}(\theta^*_{t+N}, \theta'_{t+M}), \tag{7}$$

where D measures the difference between the model parameter θ^*_{t+N} trained with the original graph $\mathcal{G}$ for N steps, and the model parameter θ'_{t+M} trained with the condensed graph $\mathcal{G}'$ for M steps from the same starting point of model parameter θ_t. Generally, θ^*_{t+N} and θ_t can be generated offline.

Eigenbasis matching [18] methods align condensed graphs and original graphs in the spectral domain via eigenbasis:

$$\mathcal{L}_e = \sum_{k=1}^{K} \left\| X^\top u_k u_k^\top X - X'^\top u'_k u'^\top_k X' \right\|_F^2, \tag{8}$$

where $u_k u_k^\top$ and $u'_k u'^\top_k$ are the subspaces induced by the k-th eigenvector in the real and synthetic graphs. A discrimination constraint to preserve the category-level information:

$$\mathcal{L}_d = \sum_{i=1}^{C} \left(1 - \frac{H_i^\top \cdot H'_i}{\|H_i\| \|H'_i\|}\right) + \sum_{\substack{i,j=1 \\ i \neq j}}^{C} \frac{H_i^\top \cdot H'_j}{\|H_i\| \|H'_j\|}, \tag{9}$$

where $H = Y^\top A X$ and $H' = Y'^\top \sum_{k=1}^{K} (1 - \lambda_k) u'_k u'^\top_k$. An additional regularisation is used to constrain the representation space:

$$\mathcal{L}_o = \left\| U'^\top_K U'_K - I_K \right\|_F^2. \tag{10}$$

The overall matching loss function of GDEM is formulated as the weighted sum of three regularisation terms:

$$\mathcal{L}_{\text{match}} = \alpha \mathcal{L}_e + \beta \mathcal{L}_d + \gamma \mathcal{L}_o. \tag{11}$$

A.2 Graph Condensation Settings

Transductive and inductive are two settings for graph dataset condensation, as introduced by [14]. The differences between these settings are illustrated in Figure 3. In the transductive setting, test nodes and their induced edges are available during training, whereas in the inductive setting, these nodes and edges are unavailable. In the testing phase of the inductive setting, the availability of labelled nodes and their induced edges is conditional. This paper assumes that labelled nodes are unavailable during the test phase in the inductive setting.

B Experiment Extensions

Hardware: One NVIDIA L40 (42GB) or one NVIDIA V100 (32GB) GPU are used for experiment.

Graph learning packages: `GCondenser` refines baseline methods to support both popular graph learning packages, DGL [3] and PyG [4], for convenience.

B.1 Condensed Graph Initialisation

Different initialisation strategies for the condensed graphs also affect the optimisation and output quality of GC methods. Stable

[3] https://www.dgl.ai/
[4] https://pyg.org/

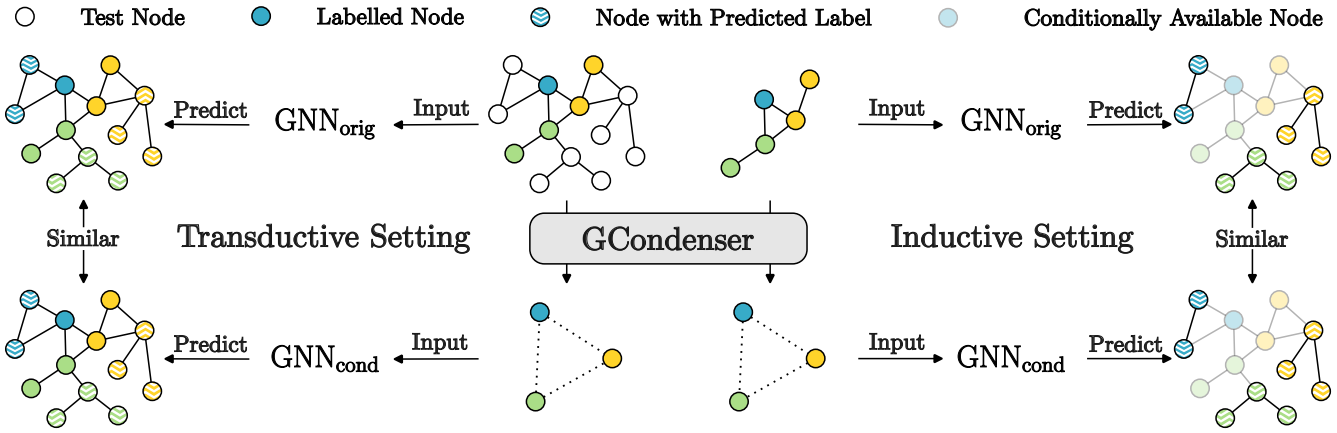


Figure 3: `GCondenser` for graph condensation with transductive (left) and inductive (right) settings.

Table 4: Condensation of Arxiv into 90 nodes with different condensed graph initialisation.

Label	Feature	GCond	DosCond	DM
Balanced	Random Noise	57.4	56.1	62.4
	Random Subgraph	56.2	55.2	62.9
	k-Center	57.6	57.2	62.7
Proportional	Random Noise	58.4	60.4	64.5
	Random Subgraph	58.4	59.4	64.1
	k-Center	57.9	59.3	63.7

optimisation and consistent condensed graph quality across different initialisations are desired for practical applications. Table 4 presents the performance of three methods (GCond, DosCond, and DM) on the Arxiv dataset for two label distributions (balanced and proportional to original) and three feature initialisation methods (random noise, random subgraph, and k-Center graph). The results show that maintaining the original label distribution leads to better performance than using a balanced label distribution. This is because, for graph data, there is generally a class imbalance issue with a large number of classes (40 classes for Arxiv). Different label initialisations directly constrain the node budget for each class. For instance, Figure 6 in Appendix C.3 illustrates how class sizes vary in the condensed graph under different label initialisations. Keeping the proportion of labels in the condensed graph can largely maintain the performance of the classes with more nodes. While for the feature initialisation, different strategies achieve comparable results across different GC methods.

B.2 Impact of Different Validators

The validator selection for the graph condensation process is essential. The validator can select a convincing graph during each condensation process and guide the hyperparameter sampler to quickly and accurately find the better hyperparameter combination. For the reliable quality of the selected graph and the effectiveness of hyperparameter search, a performance consistency of the validator on the validation set and test set is expected. GCN, SGC and GNTK as validators can serve the graph condensation process to find the optimal graph that optimises the validator to perform best on the validation set. Table 5 shows the performance of the GCN trained with different validators and GC methods and the relative convergence time compared with GCN. The results indicate that GNN validators (e.g., GCN and SGC) are effective in assessing the quality of the condensed graph. GCN produces better results but requires more time, while SGC, which caches the aggregated features across the graph, requires less time but sacrifices some performance. GNTK is the most efficient validator, but it is not as reliable compared to GNN validators.

B.3 Cross-architecture Transferability

To test the cross-architecture transferability of optimal condensed graphs from Section 4.2 After obtaining the selected optimal condensed graph. `GCondenser` can use the condensed graph to train different GNNs and evaluate the performance of the GNNs on the test set. The benchmark provides a set of architectures, including MLP, SGC [27], GCN [15], GAT [25], ChebNet [1], SAGE [10], and APPNP [16]. It is worth noting that for MLP evaluation, graph structure information is avoided in both the training and inference phases. Figure 4 shows the cross-architecture transferability of the 30-node graph condensed from the CiteSeer dataset by different GC methods. The results show that the DosCondX and GCondX methods with the GCN backbone have better average performance. More cross-architecture evaluations with different datasets are shown in Figure 14 ∼ 19.

B.4 Continual Graph Learning

To explore the effectiveness of different graph condensation methods in downstream tasks, continual graph learning (CGL) is employed for evaluating the condensed graphs. Three representative methods are selected for comparison: GCond for multi-step matching with structure learning, DosCond for one-step matching with structures learning, and DM for one-step matching without structure learning. These three methods are compared against the random subgraph method, the k-Center graph method, and the whole

Table 5: Performance of different validators on Cora, with a budget of 35-node and GCN as the backbone. The test accuracy (%) and the relative condensation time to the GCN validator are shown.

Validator	GCond	GCondX	DosCond	DosCondX	SGDD	GCDM	GCDMX	DM	SFGC	Avg.
GCN	73.8 (1.0x)	80.6 (1.0x)	78.3 (1.0x)	80.7 (1.0x)	78.7 (1.0x)	79.4 (1.0x)	78.9 (1.0x)	80.2 (1.0x)	79.6 (1.0x)	78.9 (1.0x)
SGC	67.6 (0.5x)	80.7 (0.7x)	78.2 (0.8x)	81.1 (0.9x)	64.2 (0.7x)	80.0 (1.1x)	78.5 (1.0x)	80.4 (1.0x)	79.8 (1.0x)	76.7 (0.9x)
GNTK	51.8 (0.2x)	79.7 (0.2x)	78.7 (0.8x)	78.8 (0.6x)	73.3 (0.6x)	57.6 (1.1x)	79.5 (0.6x)	78.0 (0.2x)	79.3 (0.9x)	73.0 (0.6x)

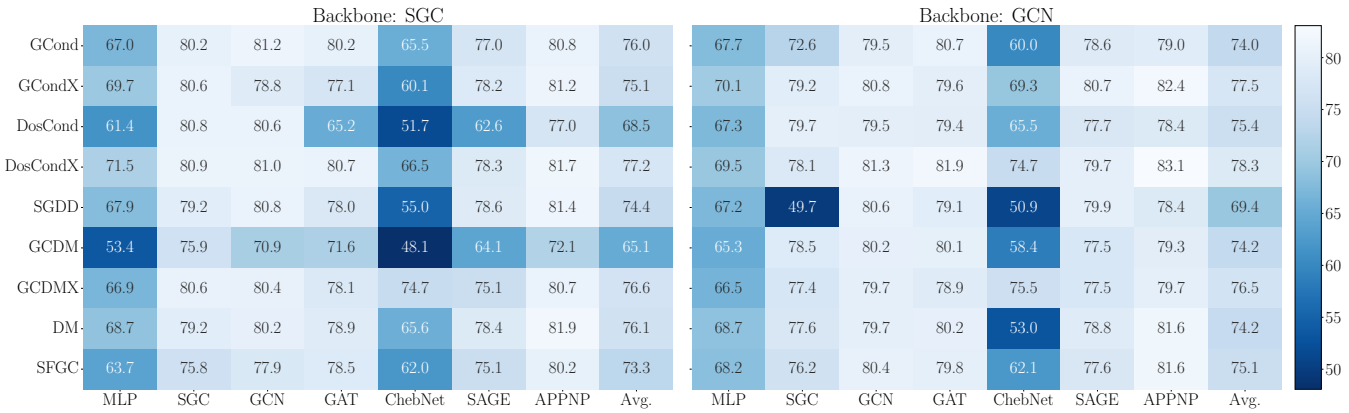


Figure 4: Transferability of condensed graphs for Cora with budget 35. More results in Appendix D.2.

graph upper-bound. It is worth noting that SFGC, as an offline algorithm, is not suitable to apply in continual graph learning, where online updates are required. The Condense and Train (CaT) framework [19] under the class incremental learning (class-IL) setting is applied. CiteSeer, Cora, Arxiv, Products, Flickr, and Reddit datasets are divided into a series of incoming subgraphs, each with two new classes. If a subgraph contains only one class, this subgraph should be removed from the streaming data. PubMed is not evaluated as it only contains three classes. Offline methods, such as SFGC, GDEM, and GEOM, are also not evaluated. Due to the running time of SGDD, a large number of tasks would be time-consuming. Therefore, SGDD is not evaluated in the CGL.

Figure 5 shows that in smaller datasets like CiteSeer and Cora, distribution-matching methods perform better due to the good adaptability of the condensed graphs to the continually learned model. DosCondX can even match the performance of training on the whole graph in long series tasks. However, most methods failed on the Flickr dataset, indicating that this dataset poses challenges for maintaining historical knowledge on the condensed graphs in the CGL setting.

C Implementation Details

C.1 Dataset Preprocessing

Row feature normalisation is applied to the CiteSeer, Cora, and PubMed datasets, whilst standardisation is utilised on the Arxiv, Flickr, and Reddit datasets. The Products dataset continues to use the features processed by OGB [12].

C.2 Graph Initialisation

The original label distribution and k-Center graph are utilised for initialisation by default. Different strategies for initialising the condensed graph are discussed in Section B.1.

C.3 Label Distribution

`GCondenser` provides two label distribution strategies: original and balanced. The original label distribution ensures that the label distribution of the synthetic graph closely matches that of the original graph, while the balanced label distribution assigns nodes uniformly to each class. Figure 6 visualises different label distributions of condensed Arxiv dataset with a 90-node budget.

C.4 Reproducibility

Condensation: Each epoch starts with the initialisation of a backbone model within a nested loop structure. The outer loop updates the condensed graph. Subsequently, the process progresses to the inner loop, where the backbone model is continuously trained using the updated condensed graph. If a structure generator (such as GCond, DosCond, SGDD, or GCDM) is employed, it is initially optimised over several epochs, followed by a few epochs dedicated to feature updates. This iterative pattern of alternating between structural and feature updates continues. For SFGC, the backbone training follows an offline style, thereby removing the need for a nested loop. Based on the condensation framework and the methodologies of various methods, hyperparameter search spaces are clearly predefined in Tables 6 and 7.

Validation: The validation phase employs the same model architecture as used in the condensation process (e.g., GCN or SGC).

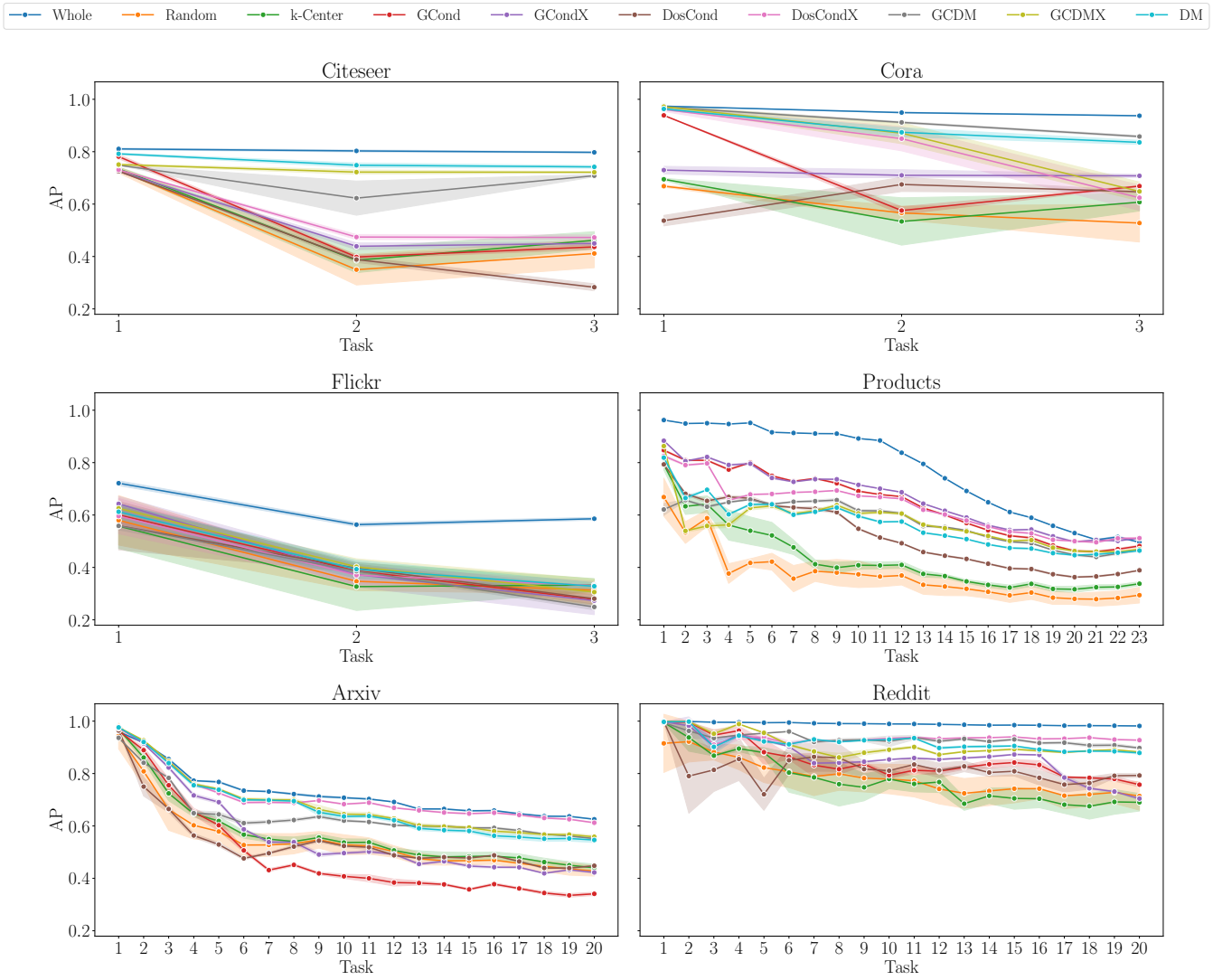


Figure 5: GC methods on baseline datasets under the continual graph learning setting.

Every 10 epochs, a validator is trained from scratch using the condensed graph and assessed against the validation set of the original graph. After 200-epoch training and validating, the best validation accuracy achieved is recorded as the validation score. The condensation process is terminated early if no improvement in validation scores is observed within five validation steps (equivalent to 50 condensation epochs); otherwise, the process continues until reaching a total of 1000 epochs.

Overall test: After condensation, the optimal condensed graphs are loaded for testing. The test model remains the same as the backbone model:

- GCN: Number of layers: 2; Hidden dimension: 256; Dropout rate: 0.5
- SGC: Number of the message passing hop (k): 2

The models are trained using the condensed graphs until convergence, employing an Adam optimiser with a weight decay of 0.5 and a learning rate of 0.01. The Bayesian hyperparameter sampler quickly identifies the optimal range for the hyperparameter combination, as illustrated in Figure 7.

C.5 Backbone Models

Besides GCN and SGC, which are used as condensation backbones and validators, the cross-architecture experiment also employs other backbone models, including MLP, GAT, ChebNet, SAGE, and APPNP:

- MLP: Number of layers: 2; Hidden dimension: 256; Dropout rate: 0.5
- GAT: Number of layers: 2; Hidden dimension: 256; Number of attention heads:8; Dropout rate: 0.5; Graph sparse threshold: 0.5

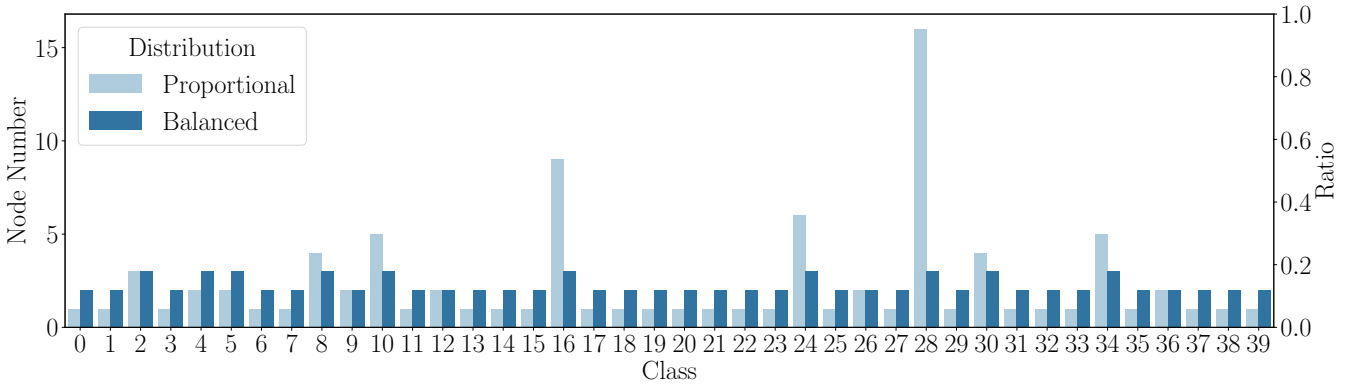


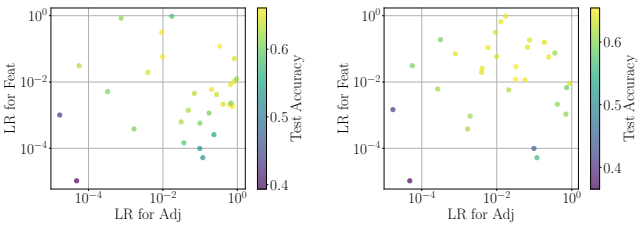
Figure 6: Comparison of the class distribution between the original and balanced label initialisation strategies with respect to the number of nodes and the class ratio in each class, using a 90-node condensed Arxiv dataset as an example.

Table 6: Predefined hyperparameter search spaces for gradient and distribution matching methods.

	GCond, SGDD, GCDM	GCondX, DosCondX	DosCond	DosCondX, DM
lr for adj	log_uniform(1e-6,1.0)	NaN	log_uniform(1e-6,1.0)	NaN
lr for feat	log_uniform(1e-6,1.0)	log_uniform(1e-6,1.0)	log_uniform(1e-6,1.0)	log_uniform(1e-6,1.0)
outer loop	5	5	1	1
inner loop	10	10	0	0
adj update steps	10	NaN	10	NaN
feat update steps	20	NaN	20	NaN

Table 7: Predefined hyperparameter search spaces for SFGC.

lr for feat	lr for student model	target epochs	warm-up epochs	student epochs
log_uniform(1e-6,1.0)	log_uniform(1e-3,1.0)	start:0, end:800, step:10	start:0, end:100, step:10	start:0, end:300, step:10



(a) GCond-SGC for Arxiv-90 **(b) SGDD-SGC for Arxiv-90**

Figure 7: Examples for overall experiments with Bayes hyperparameter sampler.

- ChebNet: Number of layers: 2; Hidden dimension: 256; Dropout rate: 0.5; Graph sparse threshold: 0.5
- SAGE: Number of layers: 2; Hidden dimension: 256; Dropout rate: 0.5; Aggregator type: mean; Graph sparse threshold: 0.5
- APPNP: Number of layers: 2; Hidden dimension: 256; Dropout rate: 0.5; Number of iterations (propagation): 10; Teleport probability (propagation): 0.1

C.6 Continual Graph Learning

Unlike the typical node classification problem, hyperparameter search in the CGL setting is challenging due to potential latency issues. This could explain why methods with fewer hyperparameters, such as DM, tend to perform better in the CGL setting. The hyperparameter choices of selected methods are shown in Table 8.

D More Experiments

In this section, more experiments are provided in addition to the ones in the main text, including the condensation efficiency and the cross-architecture transferability.

D.1 Condensation Efficiency

In this experiment, more results of the condensation efficiency on all datasets are provided in Figure 8 ∼ 13. Overall, distribution

Table 8: Hyperparameter choices for CGL experiment.

	GCond	DosCond	DM
CiteSeer	GCond: lr for adj: 1e-5; lr for feat: 1e-5; outer loop: 10; inner loop: 1	lr for adj: 1e-5; lr for feat: 1e-5; outer loop: 1	lr for feat: 1e-4; outer loop: 1
Arxiv	lr for adj: 1e-2; lr for feat: 1e-2; outer loop: 10; inner loop: 1	lr for adj: 1e-2; lr for feat: 1e-2; outer loop: 1	lr for feat: 1e-3; outer loop: 1

matching methods, such as DM, GCDM and GCDMX, can generally achieve a high performance with a high efficiency. While SFGC can sometimes obtain a higher performance, the stability and consistency are not ideal, especially given that SFGC requires an offline generation of expert trajectories.

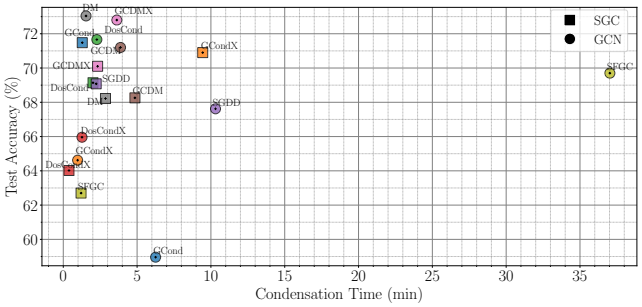


Figure 8: Test accuracy against condensation time for different GC methods on a 30-node condensed graph from the CiteSeer dataset, with backbone models GCN and SGC.

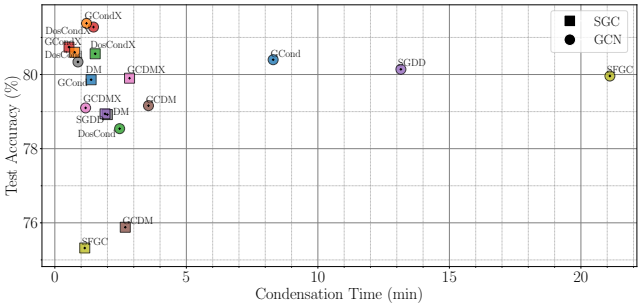


Figure 9: Test accuracy against condensation time for different GC methods on a 35-node condensed graph from the Cora dataset, with backbone models GCN and SGC.

D.2 Cross-architecture Transferability

In this experiment, more results of the cross-architecture transferability on all datasets are demonstrated in Figure 14 ∼ 19. For small-scale datasets, employing a suitable backbone model generally ensures that all baseline methods achieve comparable cross-architecture performance. However, while a condensed graph may perform well on its original architecture, its performance often deteriorates on alternative architectures when applied to large-scale datasets.

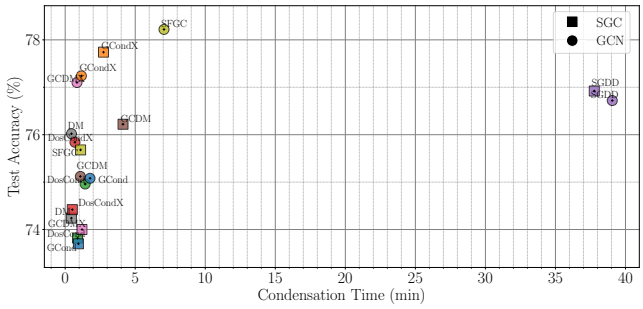


Figure 10: Test accuracy against condensation time for different GC methods on a 15-node condensed graph from the PubMed dataset, with backbone models GCN and SGC.

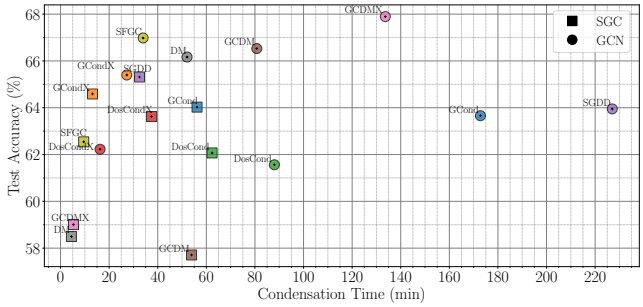


Figure 11: Test accuracy against condensation time for different GC methods on a 612-node condensed graph from the Products dataset, with backbone models GCN and SGC.

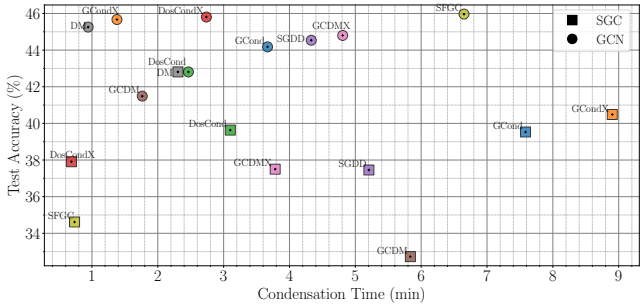


Figure 12: Test accuracy against condensation time for different GC methods on a 44-node condensed graph from the Flickr dataset, with backbone models GCN and SGC.

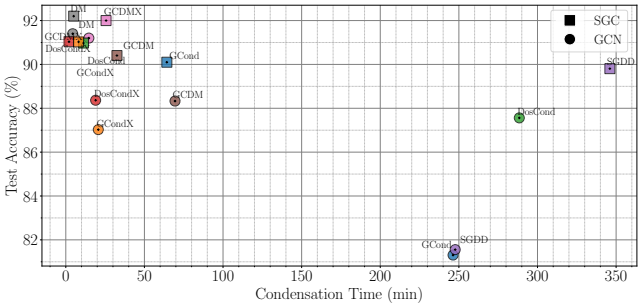


Figure 13: Test accuracy against condensation time for different GC methods on a 153-node condensed graph from the Reddit dataset, with backbone models GCN and SGC.

E Assets License

Our code base[5] is open-sourced under the MIT license. All used datasets are public: CiteSeer, Cora, PubMed, Flickr and Reddit can be obtained from DGL or PyG; ogbn-arxiv and ogbn-products are provided by OGB [6].

F Ethic Statement

Our experiments are designed and conducted in strict adherence to ethical standards and do not involve any ethical concerns. We ensure that all procedures are compliant with relevant guidelines and that there is no harm or risk to any participants or environments involved.

[5]https://github.com/superallen13/GCondenser
[6]https://ogb.stanford.edu/

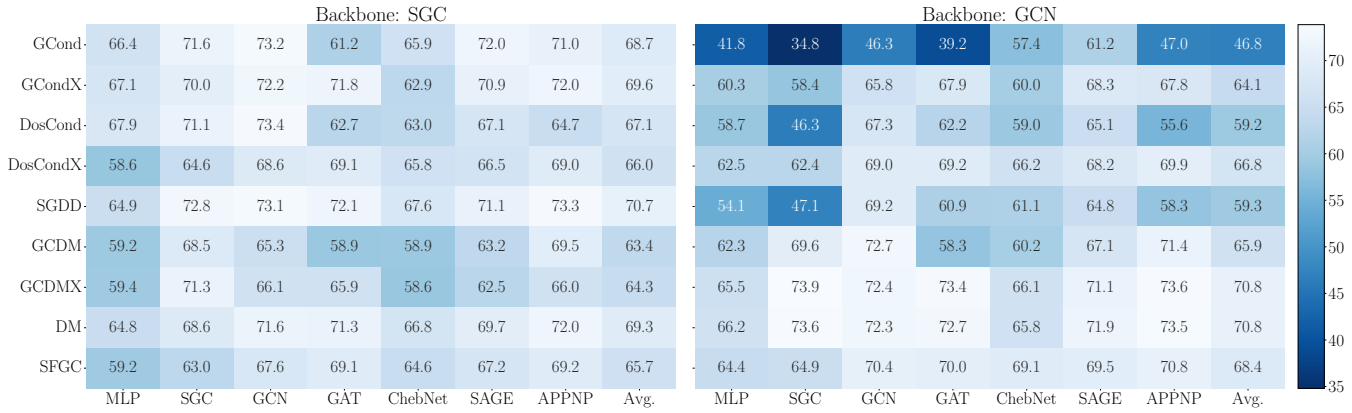


Figure 14: Transferability of condensed graphs for CiteSeer with budget 30.

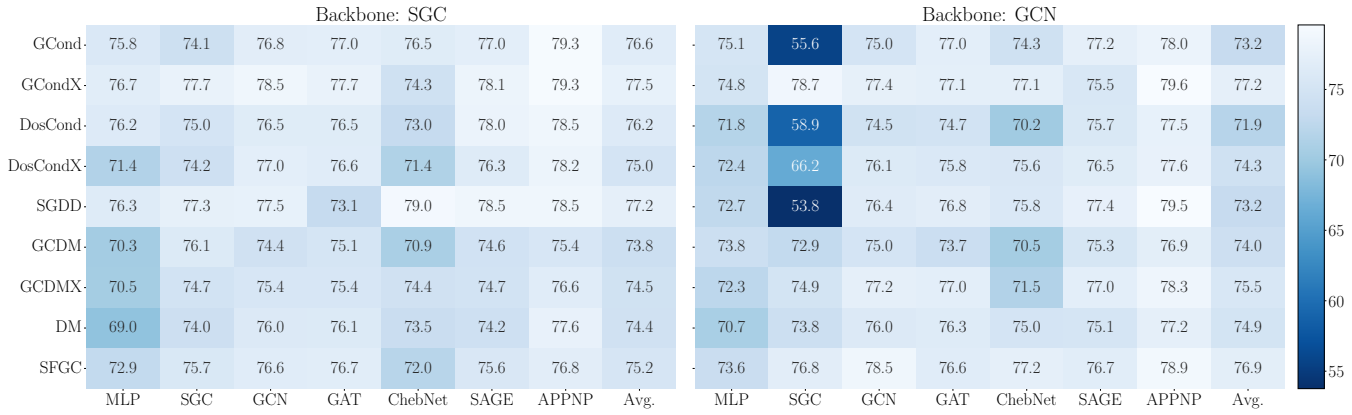


Figure 15: Transferability of condensed graphs for PubMed with budget 15.

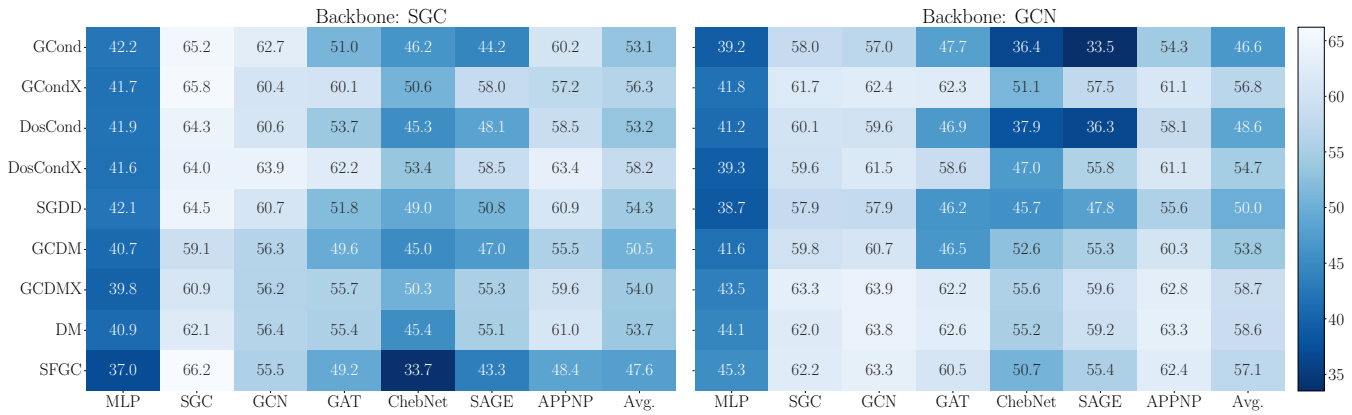


Figure 16: Transferability of condensed graphs for Arxiv with budget 90.

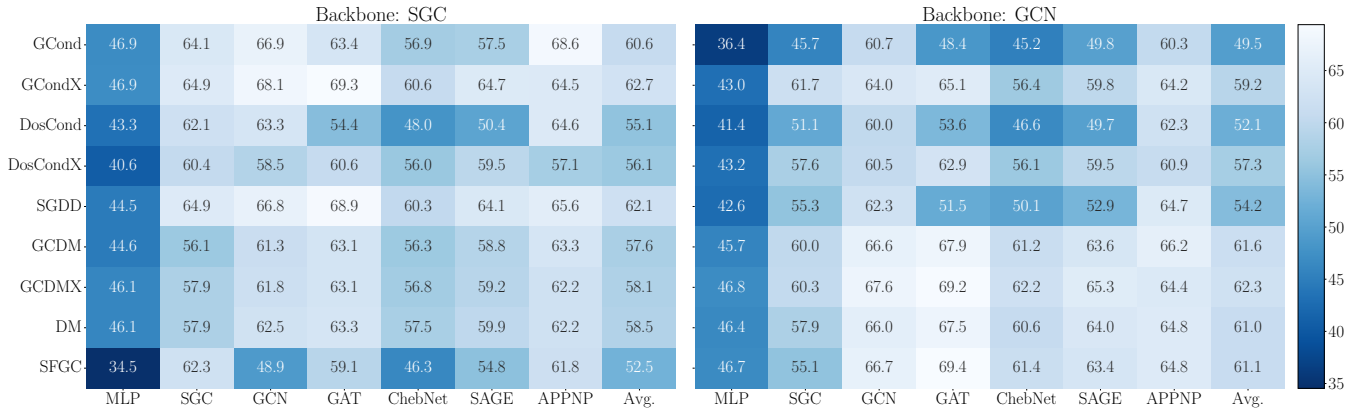


Figure 17: Transferability of condensed graphs for Products with budget 612.

Figure 18: Transferability of condensed graphs for Flickr with budget 44.

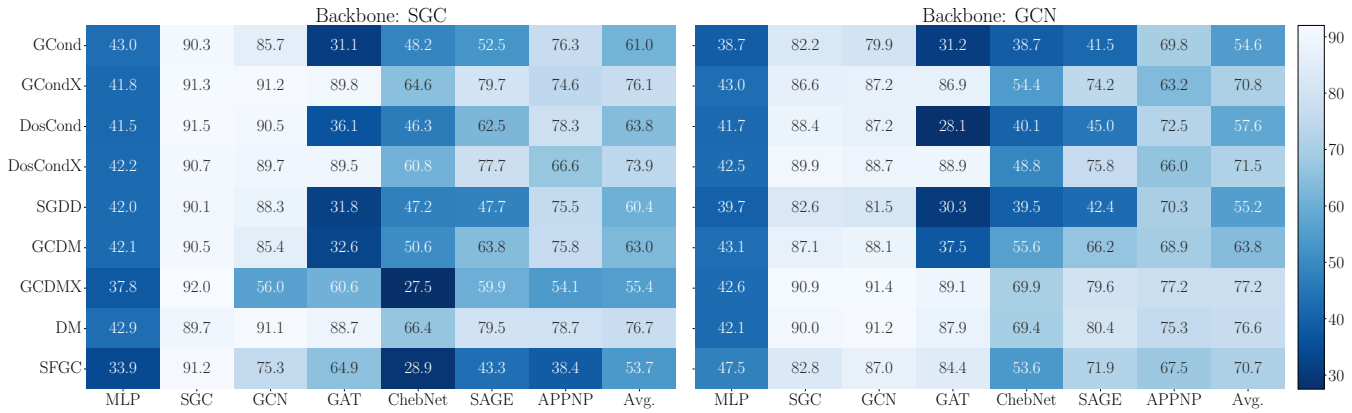


Figure 19: Transferability of condensed graphs for Reddit with budget 153.