Appendix

A Dataset Details

We use 5 node-level graph datasets (`Cora`, `Citeseer` (Kipf & Welling, 2017), `Ogbn-arixv` (Hu et al., 2020a), `Flickr` (Zeng et al., 2020), and `Reddit` (Hamilton et al., 2017)) and 5 graph-level graph datasets (`GEOM` (Axelrod & Gomez-Bombarelli, 2020), `BACE` (Wu et al., 2018), `BBBP` (Martins et al., 2012), `ClinTox` (Gayvert et al., 2016), and `SIDER` (Kuhn et al., 2016)). We further provide the statics of datasets in Table A1.

Table A1: Statistics of datasets.

Level	Dataset	# Classes / # Tasks	#Nodes / #Graphs	# Edges	# Features
Node-level	`Cora`	7	2,708	5,429	1,433
	`Citeseer`	6	3,327	4,732	3,703
	`Ogbn-Arxiv`	40	169,343	1,166,243	128
	`Flickr`	7	89,250	899,756	500
	`Reddit`	210	232,965	57,307,946	602
Graph-level	`BACE`	1	1,513	-	-
	`BBBP`	1	2,039	-	-
	`ClinTox`	2	1,478	-	-
	`Sider`	27	1,427	-	-

B Self-supervised Teacher Tasks

In the paper, we utilize different self-supervised task guided models as the way we extract the "universal knowledge" from the original dataset $\mathcal{G}$. Here, we present each model we used.

For node-level classification tasks, we follow AutoSSL (Jin et al., 2022a) and adopt five classic tasks:

- **DGI** (Velickovic et al., 2019): Maximizes the different views' representations (graph v.s. nodes).
- **CLU** (You et al., 2020b): Predicts pseudo-labels from K-means clustering on node features.
- **PAR** (You et al., 2020b): Predicts pseudo-labels from Metis graph partition (Karypis & Kumar, 1998).
- **PAIRSIM** (Jin et al., 2020): Predicts pairwise feature similarity between nodes.
- **PAIRDIS** (Peng et al., 2020): Predicts the shortest path length between nodes.

In the graph-level classification tasks, we follow WAS (Fan et al., 2024) to adopt 7 classic tasks:

- **AttrMask** (Hu et al., 2020b): Learns the regularities of node/edge attributes.
- **ContextPred** (Hu et al., 2020b): Explores graph structures by predicting the contexts.
- **EdgePred** (Hamilton et al., 2017): Predicts the connectivity of node pairs.
- **GPT-GNN** (Hu et al., 2020c): Introduces an attributed graph generation task to pre-train GNNs.
- **GraphLoG** (Xu et al., 2021): Introduces a hierarchical prototype to capture the global semantic clusters.
- **GraphCL** (You et al., 2020a): Constructs specific contrastive views of graph data.
- **InfoGraph** (Sun et al., 2019): Maximizes the mutual information between the representations of the graph and substructures.

C Proof of Theorem 1

Proof. We aim to prove that:

$$I(\mathbf{Y}_s^s; \mathbf{Y}_s^h) \leq I(\mathcal{P}^{\mathcal{T}}(\mathbf{X}, \mathbf{A}); \mathbf{Y}). \tag{A.1}$$

Since $\mathbf{Y}_s^s$ and $\mathbf{Y}_s^h$ are obtained from $\mathcal{P}^{\mathcal{T}}(\mathbf{X}, \mathbf{A})$ and $\mathbf{Y}$ through kernel ridge regression—which is a deterministic mapping, they can be expressed as:

$$\mathbf{Y}_s^s = f\left(\mathcal{P}^{\mathcal{T}}(\mathbf{X}, \mathbf{A})\right), \tag{A.2}$$

$$\mathbf{Y}_s^h = g(\mathbf{Y}), \tag{A.3}$$

where f and g are the regression functions.

According to the data processing inequality (Beaudry & Renner, 2012), applying deterministic functions to random variables does not increase mutual information. Therefore, we have:

$$I\left(\mathbf{Y}_s^s; \mathbf{Y}_s^h\right) \leq I\left(\mathcal{P}^{\mathcal{T}}(\mathbf{X}, \mathbf{A}); \mathbf{Y}\right). \tag{A.4}$$

□

D Time complexity Analysis

`ST-GCond` primarily consists of two parts: sampling sub-tasks for meta updating and involving self-supervised tasks to guide condensing. For the former, we can treat them as a composition of bi-level optimization. Following `GCond` (Jin et al., 2022c), we start with an L-layer GCN, where the large-scale graph has N nodes, the small yet informative condensed graph has m nodes, and the hidden dimension is d. The computation cost for a single task involves a forward pass through the GNN, which is $O(Lm^2d + Lmd)$, and through g_ϕ, which is $O(m^2d^2)$. The inner optimization of kernel ridge regression can be expressed as $O(Nmr^2 + Nm)$ (Wang et al., 2024). Therefore, the single task complexity is $O(Lm^2d + Lmd + m^2d^2 + Nmr^2 + Nm)$. Denoting the split of tasks as t, the complexity for the former part can be shown as $tO(Lm^2d + Lmd + m^2d^2 + Nmr^2 + Nm)$.

For the latter, the calculation process is similar to the former, although we introduce multiple self-supervised models during the condensing stage. Thanks to the benefits of the offline strategy, we only need the extra computation complexity of $kO(LEd + LNd^2)$, where k denotes the number of self-supervised tasks. Therefore, the overall complexity can be expressed as $(t+1)O(Lm^2d + Lmd + m^2d^2 + Nmr^2 + Nm) + kO(LEd + LNd^2)$. Note that t and k are not set to be too large.

To intuitively demonstrate the efficiency comparison, we present the running time (in seconds) of the proposed `ST-GCond` and `GCond` over 50 epochs on a single A100 GPU. Thanks to the efficiency of kernel ridge regression, we avoid the time-consuming triple-level optimization. As a result, our method is empirically 1.14 to 2.17 times faster than the previous `GCond` method.

Table A2: Comparison of running time of `GCond` and `ST-GCond`(in seconds).

Ogbn-arxiv	**r=0.05%**	**r=0.25%**	**r=0.5%**
`GCond` (Jin et al., 2022c)	217.18	386.71	765.12
`ST-GCond`	178.27	278.44	399.15

E Computation resource

We conduct all experiments with:

- Operating System: Ubuntu 20.04 LTS.
- CPU: Intel(R) Xeon(R) Platinum 8358 CPU@2.60GHz with 1TB DDR4 of Memory.
- GPU: NVIDIA Tesla A100 SMX4 with 40GB of Memory.
- Software: CUDA 10.1, Python 3.8.12, PyTorch (Paszke et al., 2019) 1.7.0.

F Parameter setting

In our proposed `ST-GCond`, we mainly need to handle four hyperparameters **lr**, α, β, k, the actual search space of them are:

lr	0.1, 0.01, 0.001
α	(0.0, 1.0)
β	(0.0, 1.0)
k	5

Note that the search space for α and β may change during training. For simplicity, we use 10 discrete points. The actual time consumption will depend on the Cartesian product of the individual runs.

G Algorithms

G.1 Algorithm 1

Algorithm 1 `ST-GCond`: Self-supervised and Transferable Graph Dataset Condensation

Input: Graph Dataset $\mathcal{G} = (\mathbf{X}, \mathbf{A}, \mathbf{Y})$, pre-trained teachers $\{f_1^{\mathcal{T}}(\cdot), \dots, f_K^{\mathcal{T}}(\cdot)\}$, steps T, condensation ratio r.
Output: Condensed graph dataset $\mathcal{G}_s = (\mathbf{X}_s, \mathbf{A}_s, \mathbf{Y}_s^h, \mathbf{Y}_s^s)$.
Initialize weights $\{\lambda_i = \frac{1}{K}\}_{i=1}^K$, $\mathbf{X}_s$ by selecting $r\%$ features/class, $\mathbf{Y}_s^h$ with labels.
Initialize $\mathbf{A}_s = g_\phi(\mathbf{X}_s)$, $\mathbf{Y}_s^s = \frac{1}{K}\sum \lambda_i f_i^{\mathcal{T}}(\mathbf{X}_s, \mathbf{A}_s)$.
for t = $0, \cdots, T-1$ **do**
  Initialize $\theta \sim P_\theta$.
  while not converge **do**
    $D' = 0$.
    Sample tasks $\mathcal{T}_c \sim p(\mathcal{T}_Y)$.
    for $\mathcal{T}_i$ **do**
      Sample $\mathcal{G}^{\mathcal{T}_i} \sim \mathcal{G}$, $\mathcal{G}_s^{\mathcal{T}_i} \sim \mathcal{G}_s$.
      // Meta-training
      Adapt parameters with $\mathcal{L}_{self}$ on $\mathcal{G}_s^{\mathcal{T}_i}$: $\theta_i^{'} \leftarrow \theta - \lambda_1 \nabla_\theta \mathcal{L}_{self}^{\mathcal{T}_i}(GNN_\theta, \mathcal{G}_s^{\mathcal{T}_i})$.
      // Meta-updating
      Combine representation: $\hat{\mathbf{Y}}^{\mathcal{T}_i} = \sum_{i=1}^K \lambda_i f_i^{\mathcal{T}}(\mathcal{G}^{\mathcal{T}_i})$.
      Compute $\mathcal{L}_{cls}$, $\mathcal{L}_{self}$, and $\mathcal{L}_{\mathtt{MI}}$ on $\mathcal{G}^{\mathcal{T}_i}$:
      $D' \leftarrow D' + \Big(\nabla_{\theta_i'}\mathcal{L}_{cls}^{\mathcal{T}_i}(GNN_{\theta_i'}, \mathcal{G}^{\mathcal{T}_i}) + \alpha\nabla_{\theta_i'}\mathcal{L}_{self}^{\mathcal{T}_i}(\hat{\mathbf{Y}}^{\mathcal{T}_i}, \mathcal{G}^{\mathcal{T}_i}) + \beta\nabla_{\theta_i'}\mathcal{L}_{\mathtt{MI}}^{\mathcal{T}_i}(\mathbf{Y}_s^s; \mathbf{Y}_s^h)\Big)$.
    end for
    Update $\{\lambda_i\}_{i=1}^K$, $\mathbf{X}_s$, $\mathbf{Y}_s^s$, ϕ, and θ.
  end while
end for
Generate the condensed graph: $\mathbf{A}_s = \text{ReLU}(g_\phi(\mathbf{X}_s) - \delta)$, $\mathcal{G}_s = (\mathbf{X}_s, \mathbf{A}_s, \mathbf{Y}_s^h, \mathbf{Y}_s^s)$

H More experiments

H.1 Parameter Sensitivity

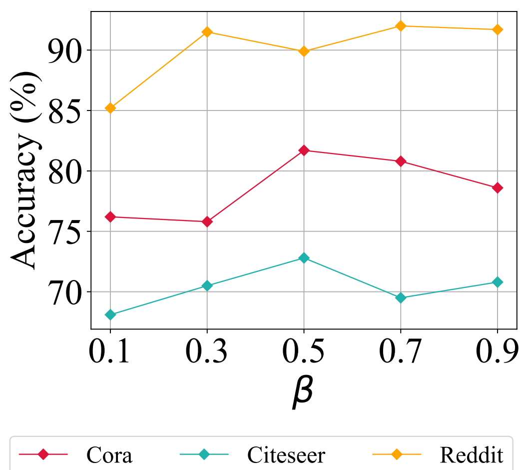

Figure A1: Sensitive of β