Appendix A. Metrics

Appendix A.1. Average Clustering Accuracy

A widely used label-required metric for evaluating clustering performance, given by:

$$\mathrm{ACC} = \max_{g \in \mathcal{G}(\mathcal{C})} \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\{y_i = g(\hat{y}_i)\} \tag{A.1}$$

where M is the number of test instances, y_i and $\hat{y}_i$ denote the ground-truth label and clustering assignment for feature $\boldsymbol{z}_i$, and $\mathcal{G}(\mathcal{C})$ is the group of permutations of $|\mathcal{C}|$ elements (this discounts the fact that the cluster indices may not be in the same order as the ground-truth labels). Permutations are optimized using the Hungarian algorithm [47]. It ranges from 0 to 1, with higher scores indicating more successful clustering.

Appendix A.2. Silhouette Coefficient

A widely used label-free metric for evaluating clustering performance, given by:

$$\mathrm{SC} = \sum_{i=1}^{M} \frac{b(\boldsymbol{z}_i) - a(\boldsymbol{z}_i)}{\max\{a(\boldsymbol{z}_i), b(\boldsymbol{z}_i)\}} \tag{A.2}$$

where M is the number of test instances, $a(\boldsymbol{z}_i)$ is the average distance between $\boldsymbol{z}_i$ and all other features within the same cluster, and $b(\boldsymbol{z}_i)$ is the smallest average distance of $\boldsymbol{z}_i$ to all features in any other cluster (of which $\boldsymbol{z}_i$ is not a member). It ranges from -1 to 1, with higher scores indicating more successful clustering.

Appendix A.3. Accuracy

A widely used metric for evaluating closed set recognition (CSR) performance, given by:

$$\text{Acc} = \frac{1}{M}\sum_{i=1}^{M}\mathbb{1}\{y_i = \hat{y}_i\} \tag{A.3}$$

where M is the number of test instances, y_i and $\hat{y}_i$ are the ground-truth and predicted labels. We adopt it to show the performance degradation of the same test set at different phases. It ranges from 0 to 1, with higher scores indicating more successful CSL. The more severe the degradation, the more the IL fails.

Appendix A.4. Harmonic Normalized Accuracy

A widely used metric for evaluating OSR performance, given by:

$$\text{HNA} = \begin{cases} 0, & if\ \text{AKS} = 0\ or\ \text{AUS} = 0 \\ \frac{2}{\left(\frac{1}{\text{AKS}} + \frac{1}{\text{AUS}}\right)}, & otherwise \end{cases} \tag{A.4}$$

where AKS and AUS are the accuracy of known and unknown classes calculated by Eq. A.3, respectively. It harmonizes AKS and AUS and ranges from 0 to 1, with higher scores indicating more successful OSR.

Appendix A.5. Final Average Accuracy

A widely used metric for evaluating final IL performance, given by:

$$\text{FAA} = \frac{1}{T}\sum_{i=0}^{T-1}\text{Acc}_i^{T-1} \tag{A.5}$$

where T is the total number of tasks, Acc_i^t is the accuracy of the model trained at the t^{th} phase on the i^{th} task calculated by Eq. A.3. It ranges from 0 to 1, with higher scores indicating more successful IL.

Appendix A.6. Final Forgetting

A widely used metric for evaluating the degree of IL degradation, given by:

$$\mathrm{FF} = \frac{1}{T-1}\sum_{i=0}^{T-2} \mathsf{f}_i, \text{ s.t. } \mathsf{f}_i = \max_{t\in\{0,\cdots,T-2\}} \mathrm{Acc}_i^t - \mathrm{Acc}_i^{T-1} \tag{A.6}$$

where f_i is the maximum discrepancy in performance for the i^{th} task. It ranges from -1 to 1, with higher scores indicating more failed IL.

Appendix B. Schematic and pseudocode of OpenGCD

Appendix B.1. Schematic

The schematic of the formulated OpenGCD is shown in Fig. B.3, where (i) and (j) are flowcharts of the proposed solution and (a)-(h) are descriptions of each component.

Appendix B.2. Pseudocode

Algorithm 1 provides the procedure for OpenGCD. Algorithms 2-4 provide the procedure of each component in OpenGCD.

Algorithm 1: Procedure of OpenGCD.

Input: Labeled data: $\mathcal{X}_0^l = \{\boldsymbol{x}_i^l, y_i^l\}_{i=1}^{N_0}$; unlabeled data: $\left\{\mathcal{X}_t^u = \{\boldsymbol{x}_i^u\}_{i=1}^{M_t}\right\}_{t=0}^{T-1}$; buffer: $\mathcal{M}_r$; feature extractor: f; classifier: φ; regulatory factor: α; Maximum total number of classes: $|\mathcal{C}_t^{\max}|$.

for *t in range*(T) **do**

- Get labeled feature embeddings $\mathcal{Z}_t^l = \{\boldsymbol{z}_i^l, y_i^l\}_{i=1}^{N_t}$ via $f(\mathcal{X}_t^l)$
- $\mathcal{M}_r = \texttt{ExemplarStorage}(\mathcal{Z}_t^l, \mathcal{M}_r)$
- Get unlabeled feature embeddings $\mathcal{Z}_t^u = \{\boldsymbol{z}_i^u\}_{i=1}^{M_t}$ via $f(\mathcal{X}_t^u)$
- $\mathcal{Z}_t^0 = \texttt{ClassifierCalibrationOSR}(\mathcal{Z}_t^u, \mathcal{M}_r, \varphi, \alpha)$
- $\mathcal{X}_t^n = \texttt{AssistingManualAnnotationGCD}(\mathcal{Z}_t^0, \mathcal{M}_r, |\mathcal{C}_t^{\max}|)$
- Get labeled exemplars $\mathcal{E}_t = \{\boldsymbol{z}_i^e, y_i^e\}_{i=1}^{N_0}$ from $\mathcal{M}_r$
- Pick the instance set $\mathcal{X}_t^e$ corresponding to $\mathcal{E}_t$
- Get new labeled instance set $\mathcal{X}_{t+1}^l = \{\boldsymbol{x}_i^l, y_i^l\}_{i=1}^{N_{t+1}}$ by concatenating $\mathcal{X}_t^e$ with $\mathcal{X}_t^n$

end

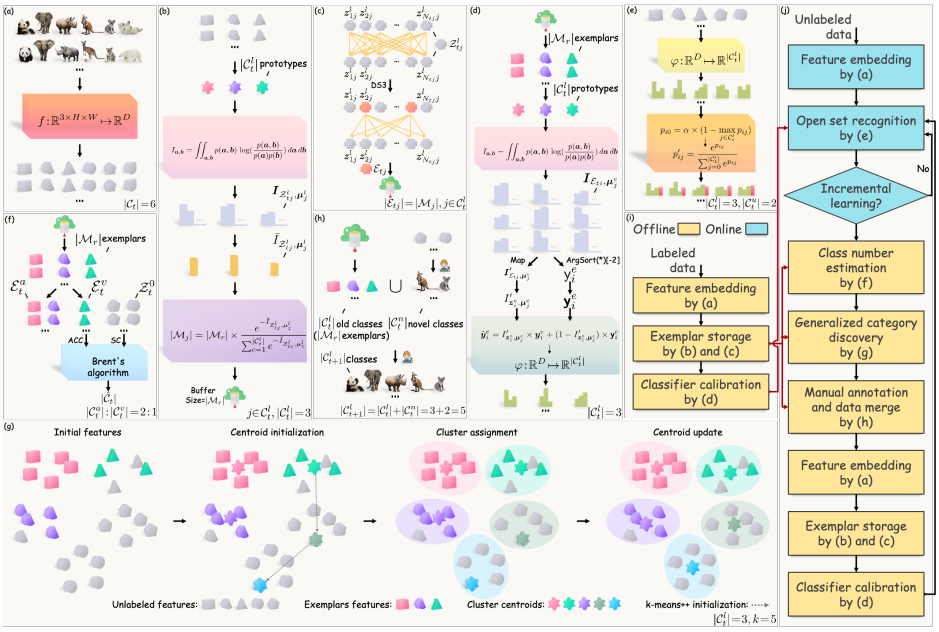


Figure B.3: Schematic of OpenGCD. (a) Feature embedding. No matter online or offline instance, its feature embedding should be obtained by the frozen feature extractor first (details c.f. Sec. 3.2 of the main paper). (b) MI-based memory allocation. Allocate a memory budget of size $|\mathcal{M}_j|$ for known class j ($j \in \mathcal{C}_t^l$) based on MI. (details c.f. Sec. 3.3.1). (c) Exemplar selection. For the known class j ($j \in \mathcal{C}_t^l$), $|\mathcal{M}_j|$ exemplars are selected by the DS3 algorithm to represent the original feature subset $\mathcal{Z}_{tj}^l$ (details c.f. Sec. 3.3.2). (d) Classifier calibration. The soft labels for (re)fitting the classifier are assigned to $|\mathcal{M}_r|$ exemplars based on MI (details c.f. Sec. 3.4). (e) Open set recognition. For an online feature, the uncertainty of the closed set output of the classifier serves as an estimate for the unknown (details c.f. Sec. 3.5). (f) Class number estimate. The clustering results of labeled $\mathcal{E}_t^v$ and unlabeled $\mathcal{Z}_t^0$ are evaluated with ACC and SC respectively and the optimal class number $|\hat{\mathcal{C}}_t|$ is determined by Brent's algorithm (details c.f. Sec. 3.6.3). (g) ss-k-means++ algorithm for $k=5$. The labeled exemplar set $\mathcal{E}_t$ and the rejected unlabeled feature set $\mathcal{Z}_t^0$ lie in the same feature space (Initial features). The centroids of $|\mathcal{C}_t^l|$ known classes are derived from labeled exemplars, and the centroids of the remaining $k-|\mathcal{C}_t^l|$ novel classes are initialized by k-means++ (Centroids initialization). For unlabeled features, clustering labels are assigned by identifying the nearest centroid (Cluster assignment). Centroids are updated by averaging the features in each cluster (Centroid update). Cluster assignment and Centroid update are then repeated until convergence, during which labeled exemplars are forced to follow their ground-truth labels (details c.f. Sec. 3.6.1). (h) Manual annotation and data merge. The labeler fetches the instance set $\hat{\mathcal{X}}_t^n$ and revises the label. Then, pick up the labeled instance set $\mathcal{X}_t^e$ and merge it with $\hat{\mathcal{X}}_t^n$ to get $\mathcal{X}_{t+1}^l$ (details c.f. Secs. 3.6.2). (i) Offline modeling procedure. Pre-stored exemplars and calibrated classifier are prepared for subsequent procedures. (j) OpenGCD procedure. Online instances can be input separately or in batches. Once a phase of OSR is completed, GCD and IL can be launched.

Algorithm 2: Exemplar Storage Procedure in OpenGCD.

Input: Labeled feature embeddings: $\mathcal{Z}_t^l = \{\boldsymbol{z}_i^l, y_i^l\}_{i=1}^{N_t}$; buffer: $\mathcal{M}_r$.
Output: Buffer filled with examples: $\mathcal{M}_r$.

Function `ExemplarStorage` $(\mathcal{Z}_t^l, \mathcal{M}_r)$**:**
  Determine known classes $\mathcal{C}_t^l$ in $\mathcal{Z}_t^l$
  for j *in* $\mathcal{C}_t^l$ **do**
    Get the feature $\mathcal{Z}_{tj}^l$ of the j^{th} class from $\mathcal{Z}_t^l$;
    Calculate the prototype $\boldsymbol{\mu}_j^l$ of the j^{th} class via $\frac{1}{|\mathcal{Z}_{tj}^l|}\sum_{y_i^l=j}\boldsymbol{z}_i^l$
    Get the MI vector $\boldsymbol{I}_{\mathcal{Z}_{tj}^l,\boldsymbol{\mu}_j^l}$ between $\mathcal{Z}_{tj}^l$ and $\boldsymbol{\mu}_j^l$ via Eq. 1
    Calculate the mean $\bar{I}_{\mathcal{Z}_{tj}^l,\boldsymbol{\mu}_j^l}$ of $\boldsymbol{I}_{\mathcal{Z}_{tj}^l,\boldsymbol{\mu}_j^l}$
  end
  for j *in* $\mathcal{C}_t^l$ **do**
    Allocate memory $\mathcal{M}_j$ for the j^{th} class via Eq. 2
    $\mathcal{M}_j = \varnothing$
    Get the feature $\mathcal{Z}_{tj}^l$ of the j^{th} class from $\mathcal{Z}_t^l$
    Select $|\mathcal{M}_j|$ exemplars $\mathcal{E}_{tj}$ by `DS3` $(\mathcal{Z}_{tj}^l)$ for the j^{th} class
    Put $\mathcal{E}_{tj}$ into $\mathcal{M}_j$
  end
  $\mathcal{M}_r = \mathcal{M}_1 \cup \mathcal{M}_2 \cup \cdots \cup \mathcal{M}_{|\mathcal{C}_t^l|}$
  return $\mathcal{M}_r$

Algorithm 3: Classifier Calibration and OSR Procedure in OpenGCD.

Input: Unlabeled feature embeddings: $\mathcal{Z}_t^u = \{\boldsymbol{z}_i^u\}_{i=1}^{M_t}$; buffer: $\mathcal{M}_r$; classifier: φ; regulatory factor: α.
Output: Rejected unlabeled feature embeddings: $\mathcal{Z}_t^0$.

Function `ClassifierCalibrationOSR` $(\mathcal{Z}_t^u, \mathcal{M}_r, \varphi, \alpha)$**:**
- Initialize classifier φ
- **for** j, $\mathcal{M}_j$ *in enumerate*$(\mathcal{M}_r)$ **do**
  - Get the exemplar $\mathcal{E}_{tj}$ of the j^{th} class from $\mathcal{M}_j$
  - Calculate the prototype $\boldsymbol{\mu}_j^e$ of the j^{th} class via $\frac{1}{|\mathcal{E}_{tj}|}\sum_{y_i^e=j}\boldsymbol{z}_i^e$
- **end**
- Get labeled exemplars $\mathcal{E}_t = \{\boldsymbol{z}_i^e, y_i^e\}_{i=1}^{N_0}$ from $\mathcal{M}_r$
- Get the MI matrix $\boldsymbol{I}_{\mathcal{E}_t}$ between $\mathcal{E}_t$ and $\boldsymbol{\mu}_j^e, j = 1, 2, \cdots, |\mathcal{C}_t^l|$ via Eq. 1
- Determine known classes $\mathcal{C}_t^l$ in $\mathcal{E}_t$
- **for** j *in* $\mathcal{C}_t^l$ **do**
  - Get the MI submatrix $\boldsymbol{I}_{\mathcal{E}_{tj}}$ related to $\mathcal{E}_{tj}$ from $\boldsymbol{I}_{\mathcal{E}_t}$
  - Get the MI vector $\boldsymbol{I}_{\mathcal{E}_{tj},\boldsymbol{\mu}_j^e}$ related to $\boldsymbol{\mu}_j^e$ from $\boldsymbol{I}_{\mathcal{E}_{tj}}$
  - Form the mapped MI vector $\boldsymbol{I}'_{\mathcal{E}_{tj},\boldsymbol{\mu}_j^e}$ via mapping $\boldsymbol{I}_{\mathcal{E}_{tj},\boldsymbol{\mu}_j^e}$ to $(0.5, 1]$
  - **for** $\boldsymbol{I}_{\boldsymbol{z}_i^e}$ *in* $\boldsymbol{I}_{\mathcal{E}_{tj}}$ **do**
    - Determine the class y_i^e corresponding to the remaining maximum MI by masking the j^{th} column of $\boldsymbol{I}_{\boldsymbol{z}_i^e}$
    - Get the labels $\boldsymbol{y}_i^e$ and $\mathbf{y}_i^e$ in one-hot form for the ground-truth class y_i^e and the nearest neighbor class y_i^e of $\boldsymbol{z}_i^e$
    - Get the mapped MI $I'_{\boldsymbol{z}_i^e,\boldsymbol{\mu}_j^e}$ between $\boldsymbol{z}_i^e$ and $\boldsymbol{\mu}_j^e$ from $\boldsymbol{I}'_{\mathcal{E}_{tj},\boldsymbol{\mu}_j^e}$
    - Get the softened label $\tilde{\boldsymbol{y}}_i^e$ of $\boldsymbol{z}_i^e$ via Eq. 3
  - **end**
- **end**
- Feed the soft-labeled exemplars $\mathcal{E}_t = \{\boldsymbol{z}_i^e, \tilde{\boldsymbol{y}}_i^e\}_{i=1}^{N_0}$ to the classifier φ for training
- $\mathcal{Z}_t^0 = \{\}$
- **for** $\boldsymbol{z}_i^u$ *in* $\mathcal{Z}_t^u$ **do**
  - Get the probability prediction $\boldsymbol{p}_i = \{p_{ij}\}_{j=1}^{|\mathcal{C}_t^l|}$ by feeding $\boldsymbol{z}_i^u$ into the fitted classifier φ
  - Calculate the uncertainty u_i in $\boldsymbol{p}_i$ via Eq. 4
  - Get the probability p_{i0} that $\boldsymbol{z}_i^u$ comes from the unknown via $\alpha \times u_i$
  - Recognize $\boldsymbol{z}_i^u$ via $y_i^* = \arg\max_{j \in \{0, \mathcal{C}_t^l\}} p_{ij}$
  - **if** $y_i^* == 0$ **then**
    - Append $\boldsymbol{z}_i^u$ to $\mathcal{Z}_t^0$
  - **end**
- **end**
- **return** $\mathcal{Z}_t^0$

Algorithm 4: Assisting Manual Annotation Procedure with GCD in OpenGCD.

Input: Rejected unlabeled feature embeddings: $\mathcal{Z}_t^0 = \{\boldsymbol{z}_i^0\}_{i=1}^{M_t^0}$;
Buffer: $\mathcal{M}_r$; Maximum total number of classes: $|\mathcal{C}_t^{\max}|$.

Output: Labeled novel class instances: $\mathcal{X}_t^n$.

Function `AssistingManualAnnotationGCD` $(\mathcal{Z}_t^0, \mathcal{M}_r, |\mathcal{C}_t^{\max}|)$**:**

Get labeled exemplars $\mathcal{E}_t = \{\boldsymbol{z}_i^e, y_i^e\}_{i=1}^{N_0}$ from $\mathcal{M}_r$

Split $\mathcal{E}_t$ into $\mathcal{E}_t^a$ and $\mathcal{E}_t^v$ by $|\mathcal{C}_t^a| : |\mathcal{C}_t^v| = 2:1$

Determine the optimal estimated number of classes $|\hat{\mathcal{C}}_t|$ by `Brent` $(\texttt{ss-k-means++}(k, \mathcal{E}_t^a, \mathcal{E}_t^v \cup \mathcal{Z}_t^0), |\mathcal{C}_t^{\max}|)$ `// Bounded by` $(|\mathcal{C}_t^a|, |\mathcal{C}_t^{\max}|]$`, supervised by` $\mathcal{E}_t^a$`, aimed at maximizing ACC on` $\mathcal{E}_t^v$ `+ SC on` $\mathcal{Z}_t^0$

Get $\hat{\mathcal{Z}}_t^n = \{\boldsymbol{z}_i^n, \hat{y}_i^n\}_{i=1}^{\hat{M}_t^n}$ by `ss-k-means++` $(|\hat{\mathcal{C}}_t|, \mathcal{E}_t, \mathcal{Z}_t^0)$ `// supervised by` $\mathcal{E}_t$`,` $k = |\hat{\mathcal{C}}_t|$

Pick the instance set $\hat{\mathcal{X}}_t^n$ corresponding to $\hat{\mathcal{Z}}_t^n$

Get $\mathcal{X}_t^n = \{\boldsymbol{x}_i^n, y_i^n\}_{i=1}^{M_t^n}$ by manually revision and label $\hat{\mathcal{X}}_t^n$

return $\mathcal{X}_t^n$

Appendix C. Ablation Study

Table C.3: Ablation study of OpenGCD.

		CIFAR100						ImageNet-100						CUB					
		IL			OSR	GCD		IL			OSR	GCD		IL			OSR	GCD	
Phase	Method	Acc			HNA	HCA		Acc			HNA	HCA		Acc			HNA	HCA	
		Ts_1			Ts_1	$Ts_{1\text{-}2}$+Tr_2	$Ts_{1\text{-}2}$+Tr_2	Ts_1			Ts_1	$Ts_{1\text{-}2}$+Tr_2	$Ts_{1\text{-}2}$+Tr_2	Ts_1			Ts_1	$Ts_{1\text{-}2}$+Tr_2	$Ts_{1\text{-}2}$+Tr_2
	CSL	**88.3%**			**88.3%**	0%	0%	**96.0%**			**96.0%**	0%	0%	**81.1%**			**81.1%**	0%	0%
	IL-E	87.6%			87.6%	0%	0%	95.7%			95.7%	0%	0%	79.7%			79.7%	0%	0%
1st	IL-EM	88.2%			88.2%	0%	0%	95.8%			95.8%	0%	0%	81.0%			81.0%	0%	0%
	OWR-EMU	88.2%			88.2%	78.3%	7.2%	95.8%			95.8%	83.3%	9.5%	81.0%			81.0%	67.1%	2.3%
	OWR-EMUC	88.2%			88.2%	80.4%	6.7%	95.8%			95.8%	**86.6%**	9.2%	81.0%			81.0%	69.6%	4.7%
	OpenGCD	87.9%			87.9%	**80.6%**	**53.3%**	95.8%			95.8%	**86.6%**	**75.7%**	81.0%			81.0%	**69.8%**	**32.9%**
		Ts_1	Ts_2		$Ts_{1\text{-}2}$	$Ts_{1\text{-}3}$+Tr_3	$Ts_{1\text{-}3}$+Tr_3	Ts_1	Ts_2		$Ts_{1\text{-}2}$	$Ts_{1\text{-}3}$+Tr_3	$Ts_{1\text{-}3}$+Tr_3	Ts_1	Ts_2		$Ts_{1\text{-}2}$	$Ts_{1\text{-}3}$+Tr_3	$Ts_{1\text{-}3}$+Tr_3
	CSL	0%	**90.5%**		30.2%	0%	0%	0%	**94.9%**		31.6%	0%	0%	0%	**70.9%**		23.7%	0%	0%
	IL-E	82.4%	81.9%		82.2%	0%	0%	94.1%	90.1%		92.8%	0%	0%	69.8%	63.5%		67.7%	0%	0%
2nd	IL-EM	83.6%	82.0%		**83.0%**	0%	0%	94.6%	91.5%		93.6%	0%	0%	**73.1%**	64.1%		70.1%	0%	0%
	OWR-EMU	**84.4%**	79.9%		82.9%	75.5%	8.9%	**94.7%**	91.8%		**93.7%**	82.7%	9.0%	**73.1%**	61.9%		**71.5%**	57.4%	4.0%
	OWR-EMUC	**84.4%**	79.9%		82.9%	**79.6%**	8.5%	**94.7%**	91.8%		**93.7%**	**84.4%**	9.4%	**73.1%**	61.9%		**71.5%**	**58.8%**	4.1%
	OpenGCD	83.5%	80.1%		82.4%	79.5%	**54.1%**	94.4%	91.9%		93.5%	**84.4%**	**61.8%**	72.6%	59.7%		70.7%	58.7%	**10.0%**
		Ts_1	Ts_2	Ts_3	$Ts_{1\text{-}3}$	$Ts_{1\text{-}4}$+Tr_4	$Ts_{1\text{-}4}$+Tr_4	Ts_1	Ts_2	Ts_3	$Ts_{1\text{-}3}$	$Ts_{1\text{-}4}$+Tr_4	$Ts_{1\text{-}4}$+Tr_4	Ts_1	Ts_2	Ts_3	$Ts_{1\text{-}3}$	$Ts_{1\text{-}4}$+Tr_4	$Ts_{1\text{-}4}$+Tr_4
	CSL	0%	0%	**90.0%**	22.5%	0%	0%	0%	0%	**96.6%**	24.1%	0%	0%	0%	0%	**66.4%**	16.6%	0%	0%
	IL-E	76.3%	75.1%	76.4%	76.0%	0%	0%	91.9%	87.3%	84.5%	88.9%	0%	0%	62.8%	**59.2%**	50.9%	59.0%	0%	0%
3rd	IL-EM	**80.1%**	**78.9%**	77.7%	**79.2%**	0%	0%	**93.3%**	**90.2%**	87.7%	91.1%	0%	0%	**65.3%**	56.9%	55.8%	60.8%	0%	0%
	OWR-EMU	79.0%	77.1%	78.6%	78.4%	72.2%	8.8%	93.2%	90.0%	89.5%	**91.5%**	81.0%	9.1%	63.1%	52.0%	56.5%	60.8%	48.3%	2.4%
	OWR-EMUC	79.0%	77.1%	78.6%	78.4%	**74.9%**	1.1%	93.2%	90.0%	89.5%	**91.5%**	82.2%	9.4%	63.1%	52.0%	56.5%	60.8%	**50.8%**	3.8%
	OpenGCD	79.2%	76.9%	78.1%	78.3%	72.2%	**40.8%**	92.5%	90.0%	90.3%	91.3%	**83.0%**	**49.0%**	63.1%	55.7%	59.2%	**61.7%**	**50.8%**	**12.6%**
		Ts_1	Ts_2	Ts_3	Ts_4	$Ts_{1\text{-}4}$		Ts_1	Ts_2	Ts_3	Ts_4	$Ts_{1\text{-}4}$		Ts_1	Ts_2	Ts_3	Ts_4	$Ts_{1\text{-}4}$	
	CSL	0%	0%	0%	**94.6%**	18.9%		0%	0%	0%	**97.1%**	19.4%		0%	0%	0%	**76.1%**	15.2%	
	IL-E	73.3%	71.4%	72.7%	74.5%	73.0%		87.5%	84.7%	83.5%	82.5%	85.1%		54.7%	**51.5%**	49.1%	67.5%	55.5%	
4th	IL-EM	75.0%	**73.6%**	74.4%	76.7%	74.9%		90.6%	**89.6%**	85.5%	88.8%	89.0%		**62.1%**	49.8%	49.2%	73.1%	59.3%	
	OWR-EMU	**76.0%**	69.6%	72.6%	74.7%	73.7%		**90.7%**	89.3%	**89.1%**	88.7%	**89.7%**		59.9%	47.5%	**54.5%**	74.3%	**59.5%**	
	OWR-EMUC	**76.0%**	69.6%	72.6%	74.7%	73.7%		**90.7%**	89.3%	**89.1%**	88.7%	**89.7%**		59.9%	47.5%	**54.5%**	74.3%	**59.5%**	
	OpenGCD	75.4%	73.2%	**76.4%**	74.7%	**75.0%**		90.1%	88.9%	88.4%	89.8%	89.5%		59.5%	49.8%	50.2%	72.1%	58.7%	

CSL: A rare version without IL (Secs. 3.3 and 3.7), OSR (Sec. 3.4.1 and Sec. 3.5), and GCD (Sec. 3.6.1). IL-E: A medium rare version without memory allocation (Sec. 3.3.1), OSR (Sec. 3.4.1 and Sec. 3.5) and GCD (Sec. 3.6.1). IL-EM: A half-baked version without OSR (Sec. 3.4.1 and Sec. 3.5) and GCD (Sec. 3.6.1). OWR-EMU: A medium well version without soft label assignment (Sec. 3.4.1) and GCD (Sec. 3.6.1). OWR-EMUC: A base version without GCD (Sec. 3.6.1) capability. OpenGCD: The full version of the proposed method described in Sec. 3.

We inspect the contributions of the various components of OpenGCD under ViT in Tab. C.3.

The CSL without IL capability offers overwhelming advantages on new classes but suffers from catastrophic forgetting. IL-E allows the classifier to retain the ability to recognize old classes by replaying exemplars. The need to balance old and new classes is the main reason for the gap between its new class recognition rate and CSL. IL-EM further enhances the IL capability of the model by memory allocation. However, they lack OSR capability, which is a nightmare for online recognition systems towards the open world as they cannot detect anomalies or isolate foreign intrusions promptly. OWR-EMU

remedies this by quantifying the unknown probability. OWR-EMUC further enhances the OSR capability of the model by calibrating the classifier with soft label assignment. The reason why OWR-EMU and OWR-EMUC still score a little on HCA although they lack GCD capability lies in the fact that they classify all novel classes as unknown, which is equivalent to clustering into one class. OpenGCD saves labelers' labor by introducing GCD to refine these unknown.

Overall, all components contribute significantly, and removing any of them can result in significant performance degradation or even loss of functionality.

Appendix D. Parametric analysis

We analyze the parameters that need to be set manually in OpenGCD with control variates.

Appendix D.1. Parameter $|\mathcal{M}_r|$

In the main paper, we report results at $|\mathcal{M}_r| = N_0$ on each dataset (20k/40k/2.4k for CIFAR100/ImageNet-100/CUB). The effect of different $|\mathcal{M}_r|$ is shown in Fig. D.4. It can be found that Acc, HNA, and HCA all show continuous or fluctuating declines over time. This is not related to the dataset type and buffer size but due to the increasing difficulty of the task. Moreover, from the first column of subfigures, it can be seen that increasing $|\mathcal{M}_r|$ improves performance but at a decreasing rate. In contrast, the effect of $|\mathcal{M}_r|$ on HNA and HCA seems to be irregular, suggesting that exploring more robust strategies for estimating regulatory factor and class number re-

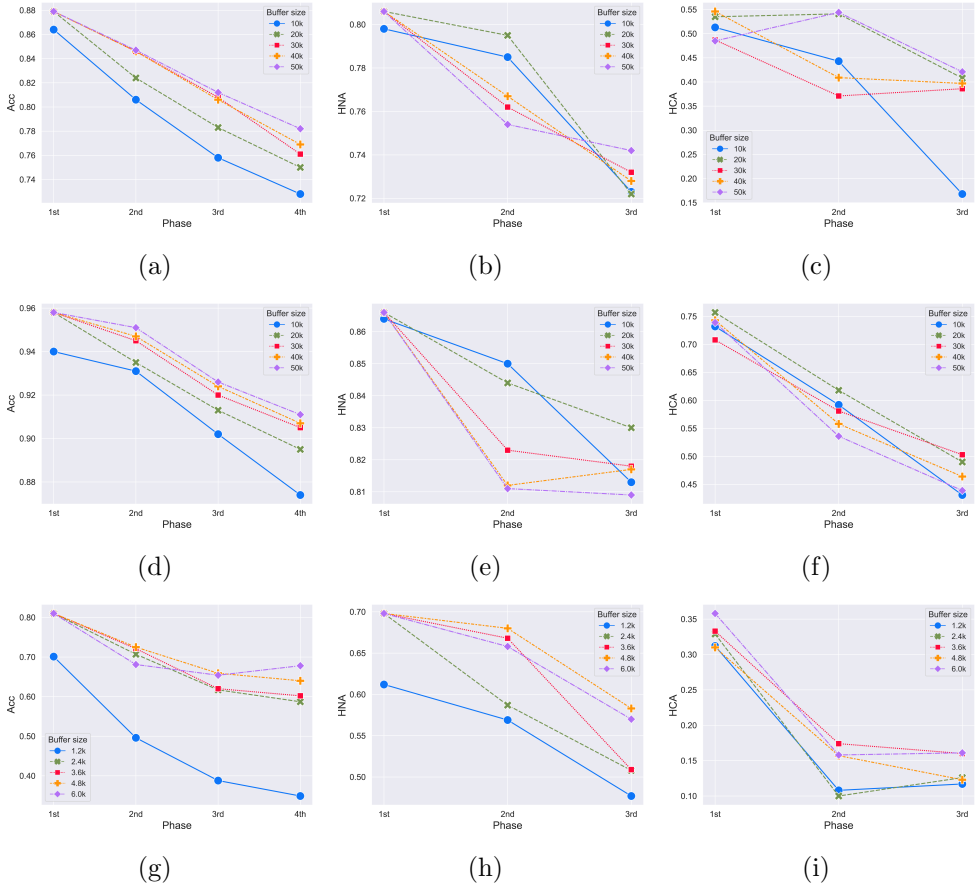
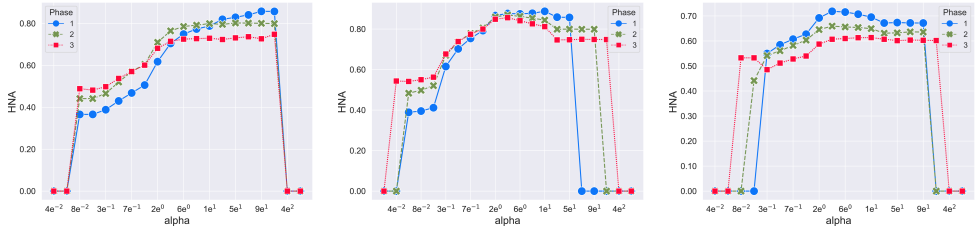


Figure D.4: Performance of OpenGCD for IL, OSR, and GCD on CIFAR100 (top), ImageNet-100 (center), and CUB (bottom) with different $|\mathcal{M}_r|$. Here, Acc is the average accuracy of all available test sets.

mains challenging. Overall, from the perspective of computational overhead, storage burden and overall performance, $|\mathcal{M}_r| = N_0$ is appropriate.

Appendix D.2. Parameter α

In the main paper, we report the results on each dataset using α determined by the open set grid search protocol ($\{1e^1, 1e^1, 1e^1\}/\{4e^0, 1e^0, 1e^0\}/\{6e^0, 8e^{-1}, 5e^{-1}\}$ for CIFAR100/ImageNet-



(a) (b) (c)

Figure D.5: Performance of OpenGCD for OSR on CIFAR100 (left), ImageNet-100 (center), and CUB (right) with different α.

100/CUB). The effect of different α is shown in Fig. D.5. It can be found that with the increase of α, HNA presents a trend of rising first and then falling. The α determined on CIFAR100 and ImageNet-100 are close to or even equal to the optimal point, but is poor on CUB. Overall, the open set grid search protocol is still recommended in the absence of a better hyperparameter optimization strategy.

Appendix E. Experiments on each subtask

We compare OpenGCD with popular OSR, GCD, IL, and CGCD baselines, respectively. Following the main paper, experiments on OSR still use CIFAR100, ImageNet-100, and CUB representing different scale datasets. Following [16, 29], experiments on GCD are conducted on three benchmark classification datasets CIFAR10, CIFAR100 and ImageNet-100. Following [19, 35, 48], three popular datasets CIFAR10, CIFAR100 and Tiny-ImageNet are used in the experiments on IL and CGCD.

Appendix E.1. Experiments on Open Set Recognition

To simulate the open set scenario, we randomly pick 30%/60%/90% of the classes from the training set of CIFAR100, ImageNet-100, and CUB as the initial known classes (openness is 45%/23%/5%), and then perform the validation of each method on the full test set containing 100/100/200 classes. As in [49], we use HNA as evaluation metric. In OSR baselines, OSNN, OpenMax and OSSVM originally fixed backbone, and to be fair we replaced them with ViT without fine-tuning under DINO. Moreover, PixMix/OpenGBPA is not network dependent, it swaps the same backbone (fixed ViT without fine-tuning under DINO) and classifier as ours.

For OSR, the comparison results of OpenGCD and other baselines are listed in Tab. E.4. In order to demonstrate the online running efficiency of various methods, we also included the time spent by each method in the entire testing process in Tab. E.4. It can be found that OSNN as a thresholding method significantly outperforms the other baselines in terms of speed and the performance is comparable, which shows that it is simple and effective. However, it is impotent in scenarios with a large number of classes, as it needs to compute the distance more times and the sensitivity to the threshold increases dramatically as the number of classes increases. Although PixMix also works on the output of closed set classifier like OSNN, the number of classes has a negligible effect on its efficiency and sensitivity, thus ensuring its speed and performance. The proposed method also belongs to the calibration family like PixMix and is therefore also satisfactory in terms of performance and speed. However, OpenGCD does not destroy the original input information as PixMix does. OpenMax, OSSVM, and OpenGBPA are

Table E.4: Comparison of OpenGCD with Other OSR Methods.

	CIFAR100						ImageNet-100						CUB					
Openness	**45%**		**23%**		**5%**		**45%**		**23%**		**2%**		**45%**		**23%**		**5%**	
Methods	**HNA**	**Time(s)**	**HNA**	**Time(s)**	**HNA**	**Time(s)**	**HNA**	**Time(s)**	**HNA**	**Time(s)**	**HNA**	**Time(s)**	**HNA**	**Time(s)**	**HNA**	**Time(s)**	**HNA**	**Time(s)**
OSNN[49]	77.7%	**0.9**	60.4%	**2.2**	25.5%	**2.8**	83.6%	**2.0**	78.8%	**3.2**	49.4%	**5.9**	64.8%	**1.1**	57.4%	2.7	23.0%	3.7
OpenMax[50]	76.7%	5.1	68.8%	7.0	40.3%	9.3	88.8%	10.2	80.9%	17.1	68.9%	18.1	60.0%	4.8	57.3%	6.9	38.0%	9.3
PixMix[8]	71.9%	1.6	58.2%	1.6	37.7%	1.6	86.6%	9.3	80.5%	9.4	76.5%	9.9	70.5%	1.6	57.5%	1.6	21.7%	1.6
OSSVM[27]	77.7%	551.7	68.1%	2103.6	34.2%	5416.9	83.6%	2662.9	60.6%	10580.2	19.7%	22165.1	75.9%	201.1	54.9%	505.2	18.2%	873.4
OpenGBPA[51]	77.3%	6.6	68.5%	7.2	35.3%	12.1	85.3%	11.6	85.2%	15.5	68.2%	18.1	73.1%	4.9	**63.8%**	7.9	31.6%	10.4
OpenGCD	**79.6%**	2.9	**70.0%**	3.1	**60.0%**	3.0	**89.3%**	6.4	**87.2%**	5.8	**81.8%**	6.4	**81.4%**	2.2	62.5%	**2.0**	**41.7%**	**2.1**

The OpenGCD here contains only Secs. 3.2, 3.4, and 3.5 of the main paper.

less efficient because they require fitting additional evaluation models. It is worth mentioning that OpenGCD also has the same unknown probability estimation capability as OpenMax and OpenGBPA, which is more intuitive and user-friendly. Overall, our strengths over threshold-based methods (e.g., OSNN) are user-friendliness and intuition. Our strength over calibration-based methods (e.g., PixMix) is that the original input information is not destroyed. Our strength over methods based on the 1-versus-all principle and evaluation (e.g., OpenMax, OSSVM, and OpenGBPA) is computational lightweight.

Appendix E.2. Experiments on Generalized Category Discovery

To simulate the GCD scenario, we follow [12] to randomly pick 5/80/50 classes from the training set of CIFAR10/CIFAR100/ImageNet-100 as the initial labeled known classes, and then the full test set containing 10/100/100 classes as the unlabeled data. The evaluation metrics are our AKS, ANS, and HCA, which evaluate the old, new, and all classes in the test set, respectively. In GCD baselines, GCD/SimGCD/PIM used ViT under DINO (with fine-tuning), ORCA/OpenCon used ResNet-18 and -50 under SimCLR (with fine-tuning). All baselines remain intact. To be fair, we provide the performance of OpenGCD when using the same pre-training, backbone, and fine-tuning

as ORCA [16] and PIM [40], respectively.

For GCD, the comparison results of OpenGCD and other baselines are listed in Tab. E.5. In addition to performance and time overhead, we also provide a comparison of the class number estimation in Tab. E.5. It should be noted that the default number of classes for SimGCD and PIM is known and taken from Max-ACC (GCD) [12] and Max-ACC (PIM) [40], respectively. Therefore, SimGCD and PIM actually take more time than given in Tab. E.5. It can be found that under ResNet, ORCA's performance is close to ours, but the time cost is at least 1.4 times more than ours. OpenCon has a slight performance advantage, but the time loss is prohibitive. Under ViT, GCD gives acceptable performance in the least time, but using known class accuracy as the only criterion makes it underperform in class number estimation and thus lose in performance. In contrast, OpenGCD trades off a more accurate class number estimation with minimal time cost, resulting in improved performance. SimGCD and PIM are ahead in performance, but they pay no attention to the estimation of the number of classes, which is a prerequisite for the success of the algorithms, and the time cost is prohibitive, especially for SimGCD. Overall, compared with other methods, the estimation ability, prediction performance and computational cost of OpenGCD are satisfactory.

Appendix E.3. Experiments on Incremental Learning

To simulate the IL scenario, we follow [35, 48] to divide the 10/100/200 classes in CIFAR10/CIFAR100/Tiny-ImageNet into 5/20/10 disjoint tasks, each task contains 2/5/20 classes. As in [52], we use FAA and FF, given in Appendix B, as evaluation metrics. In IL baselines, MCSS used a customized

Table E.5: Comparison of OpenGCD with Other GCD Methods.

	CIFAR10					CIFAR100					ImageNet-100				
Methods	$k_{GT}=10$	AKS	ANS	HCA	Time(s)	$k_{GT}=100$	AKS	ANS	HCA	Time(s)	$k_{GT}=100$	AKS	ANS	HCA	Time(s)
ORCA[16]	**10**	86.7%	85.7%	86.2%	5010.0	114	62.6%	41.4%	49.8%	7023.7	106	88.5%	73.6%	80.4%	35493.3
OpenCon[29]	**10**	86.8%	84.3%	85.5%	7149.6	109	64.9%	44.5%	52.8%	10928.7	**99**	92.2%	**79.8%**	85.5%	45516.2
$\text{OpenGCD}_{\text{ResNet}}$	**10**	86.1%	84.4%	85.2%	3372.1	107	61.1%	50.5%	55.2%	5005.6	107	88.6%	73.7%	80.7%	11506.8
GCD[12]	9	98.1%	85.8%	91.5%	**1100.4**	**100**	81.2%	65.6%	72.6%	**1182.1**	109	93.1%	67.4%	78.2%	**6015.2**
$^{\triangledown}$SimGCD[39]	9	96.0%	74.4%	83.8%	47091.4	**100**	84.9%	**75.0%**	**79.6%**	47752.4	109	**95.3%**	77.7%	**85.6%**	210178.8
$^{\triangledown}$PIM[40]	**10**	97.4%	**93.3%**	**95.3%**	1586.5	95	**85.6%**	69.4%	76.7%	1665.6	102	**95.3%**	76.9%	85.1%	6836.7
$\text{OpenGCD}_{\text{ViT}}$	**10**	**98.6%**	91.7%	95.0%	1192.5	107	81.4%	70.7%	75.7%	1287.7	103	93.4%	73.4%	82.2%	6373.9

Downward triangle ($\triangledown$) denotes that the method defaults to the total number of classes being known, i.e., their time in the table excludes the time required to estimate k. The OpenGCD here contains only Secs. 3.2, 3.6.1, and 3.6.3 of the main paper.

network structure, and ResNet-18 was used for the remaining baselines. All baselines remain intact. To be fair, in addition to ViT without fine-tuning under DINO, we also provide the performance of OpenGCD when using the same ResNet as DER++ [48]. There is no pre-training, each phase is trained from scratch with cross-entropy on all data currently available, and ResNet (no final linear layer) is frozen to replace ViT after training.

For IL, we provide in Tab. E.6 a comparison of the results for two cases with buffer sizes of N_0 (10k/2.5k/10k for CIFAR10/CIFAR100/Tiny-ImageNet) and $2\times N_0$ (20k/5k/20k for CIFAR10/CIFAR100/Tiny-ImageNet), respectively. It can be found that the performance and speed of iCaRL, which is a pioneer of replay methods, is acceptable. The performance and speed of DER++ is poor with a large number of classes, whereas CoPE is affected not only by the number of classes but also by the volume of data. X-DER, an improved version of DER++, improves performance significantly, but the computational overhead grows prohibitive. The lead of OpenGCD compared to methods also under ResNet shows that our strength does not stem from ViT under DINO. MCSS earns a place in the race with its customized network structure and focus on multiple criteria. In contrast, OpenGCD captures the key to instance retention, i.e., diversity, and complements it with a small cost

Table E.6: Comparison of OpenGCD with Other IL Methods.

$\lvert\mathcal{M}_r\rvert$	CIFAR10						CIFAR100						Tiny-ImageNet					
	10k			20k			2.5k			5k			10k			20k		
Methods	FAA	FF↓	Time(s)	FAA	FF↓	Time(s)	FAA	FF↓	Time(s)	FAA	FF↓	Time(s)	FAA	FF↓	Time(s)	FAA	FF↓	Time(s)
iCaRL[22]	94.2%	5.3%	5693.0	95.4%	3.5%	7802.2	53.6%	45.0%	6629.0	59.4%	38.3%	9423.8	57.9%	30.3%	12373.0	59.1%	29.0%	17332.4
DER++ [48]	92.8%	6.3%	6870.9	94.3%	4.7%	7223.6	52.8%	34.6%	6995.7	57.2%	29.6%	7019.6	43.7%	36.3%	42215.4	50.3%	29.3%	42315.6
CoPE [23]	93.4%	6.1%	5440.1	95.1%	3.0%	6643.4	27.1%	70.5%	5668.7	43.1%	49.5%	8042.2	38.6%	34.6%	11239.0	40.3%	25.9%	15468.6
X-DER [52]	95.8%	3.2%	25176.8	96.2%	2.8%	43068.5	58.3%	28.7%	25870.6	60.7%	28.1%	27556.4	56.0%	23.7%	172650.0	57.2%	21.5%	242279.0
$\text{OpenGCD}_{\text{ResNet}}$	95.0%	**2.3%**	5037.9	96.1%	2.2%	7548.0	63.7%	18.6%	6390.9	70.0%	17.8%	9753.2	62.1%	13.8%	8452.23	64.1%	11.5%	16417.9
MCSS [35]	**95.9%**	2.8%	5894.7	**96.4%**	2.2%	8011.5	66.1%	32.6%	6838.7	72.1%	24.9%	9870.2	65.5%	23.6%	12416.1	68.5%	18.6%	18175.5
$\text{OpenGCD}_{\text{ViT}}$	95.7%	**2.3%**	**3396.0**	96.2%	**2.0%**	**6001.2**	**69.2%**	**14.9%**	**3439.2**	**74.1%**	**12.4%**	**6628.1**	**66.1%**	**11.3%**	**6578.3**	**70.7%**	**9.0%**	**12544.4**

Downward arrow (↓) denotes that the lower the score, the better the performance. The OpenGCD here contains only Secs. 3.2, 3.3, 3.4.2, and 3.7 of the main paper.

of memory allocation yielding a win-win situation in terms of performance and speed. Overall, our strength over replay methods (e.g., iCaRL, CoPE, MCSS) is the rational allocation of memory resources. Our strength over regularization methods (e.g., DER++, X-DER) is computational lightness.

Appendix E.4. Experiments on Continual Generalized Category Discovery

Given the similarity of INCD, CNCD, IGCD, and CGCD, we only select the most challenging CGCD for the comparison experiment. To simulate the CGCD scenario, we follow [19] to divide the training set of CIFAR10/CIFAR100/Tiny-ImageNet into labeled and unlabeled sets according to 8:2. Then, 7/80/150 classes are randomly picked from the labeled set as initial known classes. Afterwards, 2000/2000/3000 samples were selected from $7 + t/80 + 5t/150 + 10t$ known classes in the unlabeled set, and 3000/1500/3000 samples were selected from 1/5/10 randomly picked novel class in the unlabeled set. Therefore, there are 3/4/5 incremental steps for CIFAR10/CIFAR100/Tiny-ImageNet. The evaluation metrics are our AKS, ANS, and HCA, which evaluate the old, new, and all classes of the selected samples in the unlabeled set, respectively. As in [14], the buffer budget is 2k for all three datasets. In CGCD baselines, GCD/PA/MetaGCD used ViT under DINO (with fine-tuning), iNatIGCD/GM used ResNet-18 (with fine-

tuning under MoCo/without pre-training). To be fair, we remove the pre-training of iNatIGCD. The remaining baselines remain intact. Moreover, we provide the performance of OpenGCD when using the same pre-training and backbone as MetaGCD [19] and GM [14], respectively. $\text{OpenGCD}_{\text{ViT}}$ is fine-tuned as in [40]. $\text{OpenGCD}_{\text{ResNet}}$ is no pre-training, each phase is trained from scratch with supervised and self-supervised contrastive loss on all data currently available, discarding the final linear layer after training.

For CGCD, the comparison results of OpenGCD and other baselines are listed in Tab. E.7. In addition to performance and time overhead, we also provide a comparison of the class number estimation in Tab. E.7. It should be noted that GM employs the class number estimation protocol in GCD, i.e., Max-ACC (GCD), and MetaGCD defaults to a known true number of classes. Under ResNet, OpenGCD is close to or even ahead of GM in terms of performance and speed. This is attributed to our more comprehensive class number estimation protocol. The performance of iNatIGCD is competitive, but its time loss is large and the class number estimation is unstable. Under ViT, as concluded in Sec. Appendix E.2, GCD suffers the least time penalty, but one-sided class number estimation limits its performance. PA performs well in scenarios with few classes, but unstable class number estimation overwhelms it in scenarios with many classes, which is particularly evident in Tiny-ImageNet. This shows that the memory-efficient exemplar replay strategy, which retains only the class mean and variance, is not competent for scenarios with a large number of incremental steps. Backed by the actual number of classes, MetaGCD leads in performance, but its time loss is prohibitive. OpenGCD keeps the performance a close second with

Table E.7: Comparison of OpenGCD with Other CGCD Methods.

		CIFAR10					CIFAR100					Tiny-ImageNet				
Phase	**Method**	**Est. k**	**AKS**	**ANS**	**HCA**	**Time(s)**	**Est. k**	**AKS**	**ANS**	**HCA**	**Time(s)**	**Est. k**	**AKS**	**ANS**	**HCA**	**Time(s)**
1st	iNatIGCD[17]	10	89.0%	64.6%	74.9%	1060.8	88	53.0%	48.7%	50.8%	1504.4	**160**	69.1%	62.1%	65.4%	6227.9
	GM[14]	9	88.9%	79.4%	83.9%	344.5	**85**	69.5%	57.3%	62.8%	688.1	170	61.0%	41.7%	49.5%	2934.4
	$\text{OpenGCD}_{\text{ResNet}}$	**8**	89.2%	89.0%	89.1%	324.9	86	68.3%	53.6%	60.1%	609.7	162	68.3%	58.1%	52.3%	2713.2
	$^{\Diamond}$GCD[12]	9	92.7%	89.9%	91.3%	**126.9**	88	78.6%	66.9%	72.3%	**315.5**	168	72.8%	63.9%	68.1%	**1835.6**
	PA[20]	**8**	**98.2%**	**92.5%**	**95.3%**	1109.6	89	78.2%	64.9%	70.9%	1439.0	156	78.2%	**67.0%**	**72.2%**	6562.6
	MetaGCD[19]	-	98.1%	92.4%	95.2%	8067.0	-	**80.9%**	**69.7%**	**74.9%**	15609.4	-	**78.6%**	64.8%	71.0%	71876.9
	$\text{OpenGCD}_{\text{ViT}}$	**8**	96.4%	90.1%	93.1%	134.0	87	78.3%	69.0%	73.4%	360.5	159	72.1%	66.8%	69.3%	2108.2
2nd	iNatIGCD[17]	11	87.2%	67.1%	75.8%	1386.7	93	48.0%	41.6%	44.6%	1856.5	**172**	63.1%	54.4%	58.4%	6709.5
	GM[14]	10	88.8%	79.0%	83.6%	447.8	91	64.7%	53.1%	58.3%	780.7	176	58.4%	35.5%	44.2%	3121.5
	$\text{OpenGCD}_{\text{ResNet}}$	**9**	88.9%	88.3%	88.6%	355.2	89	63.1%	51.2%	56.5%	655.4	**168**	60.8%	51.7%	55.9%	3100.7
	$^{\Diamond}$GCD[12]	10	92.1%	85.9%	88.9%	**127.8**	91	71.3%	61.8%	66.2%	**454.6**	174	70.5%	61.3%	65.6%	**2096.3**
	PA[20]	12	**98.0%**	72.2%	83.1%	1470.5	93	68.5%	57.3%	62.4%	1760.9	**172**	71.3%	**64.5%**	**67.7%**	6836.5
	MetaGCD[19]	-	97.8%	**90.7%**	**94.1%**	11984.2	-	**74.8%**	**66.1%**	**70.2%**	20746.5	-	**72.9%**	62.5%	67.3%	104622.1
	$\text{OpenGCD}_{\text{ViT}}$	**9**	95.4%	89.9%	92.6%	158.8	**90**	72.1%	64.7%	62.3%	529.6	**172**	70.9%	63.6%	67.1%	2463.8
3rd	iNatIGCD[17]	14	79.1%	23.1%	35.8%	1716.7	99	46.8%	29.4%	36.1%	2478.4	182	60.7%	51.3%	55.6%	7164.5
	GM[14]	11	88.1%	77.8%	82.6%	553.5	94	62.5%	51.3%	56.3%	871.3	182	54.8%	32.8%	41.0%	3930.7
	$\text{OpenGCD}_{\text{ResNet}}$	**10**	88.0%	84.9%	86.4%	416.2	**95**	62.8%	51.6%	56.7%	693.1	179	59.1%	49.4%	53.8%	3602.2
	$^{\Diamond}$GCD[12]	11	91.3%	81.6%	86.2%	**159.0**	94	68.7%	55.5%	61.4%	**598.9**	**180**	68.2%	56.0%	61.6%	**2518.9**
	PA[20]	13	**96.0%**	72.8%	82.8%	1832.2	98	67.4%	54.2%	60.1%	2794.7	174	62.9%	52.3%	57.1%	7585.2
	MetaGCD[19]	-	94.3%	**83.1%**	**88.3%**	15904.9	-	**72.7%**	**65.5%**	**68.9%**	24664.1	-	**69.4%**	**58.8%**	**63.7%**	134714.5
	$\text{OpenGCD}_{\text{ViT}}$	**10**	93.5%	81.7%	87.2%	162.4	94	71.2%	61.2%	65.8%	630.6	178	67.9%	57.1%	62.0%	2825.9
4th	iNatIGCD[17]						102	42.6%	24.1%	30.8%	3274.9	196	56.0%	47.1%	51.2%	7663.8
	GM[14]						97	54.3%	42.7%	47.8%	984.5	188	50.6%	29.0%	36.9%	4610.1
	$\text{OpenGCD}_{\text{ResNet}}$						**101**	57.2%	42.9%	49.0%	890.2	**190**	56.9%	48.4%	52.3%	4129.7
	$^{\Diamond}$GCD[12]						97	59.6%	48.1%	53.2%	**786.3**	188	61.8%	52.2%	56.6%	**2886.9**
	PA[20]						103	62.7%	51.2%	56.4%	3230.0	194	57.9%	47.1%	51.9%	7981.5
	MetaGCD[19]						-	**67.1%**	**57.4%**	**61.9%**	28648.8	-	**64.1%**	**56.7%**	**60.1%**	172648.7
	$\text{OpenGCD}_{\text{ViT}}$						98	63.2%	51.1%	56.5%	843.0	192	62.8%	55.6%	59.0%	3226.5
5th	iNatIGCD[17]											206	49.4%	41.5%	45.1%	8232.4
	GM[14]											194	46.8%	21.7%	29.7%	5249.3
	$\text{OpenGCD}_{\text{ResNet}}$											**202**	50.3%	41.4%	45.4%	4816.4
	$^{\Diamond}$GCD[12]											194	56.1%	48.2%	51.9%	**3185.4**
	PA[20]											208	52.7%	41.5%	46.4%	8398.3
	MetaGCD[19]											-	**57.2%**	**51.6%**	**54.3%**	200541.6
	$\text{OpenGCD}_{\text{ViT}}$											**198**	56.4%	50.2%	53.1%	3584.8

Lozenge ($\Diamond$) denotes that the original method has only GCD capability, and we embed the proposed IL method to give it CGCD capability. Bar ($-$) denotes that the method defaults to a known number of novel classes and lacks the ability to estimate the number of classes. The OpenGCD here contains only Secs. 3.2, 3.3, 3.6.1, 3.6.3, and 3.7 of the main paper.

efficient memory allocation and excellent class number estimation. Overall, OpenGCD's overall score is the best, showing that for CGCD, a fast, stable, and accurate class number estimation is the key to victory.