DCHA: DYNAMIC COMPOSITIONAL ADAPTATION WITH HIERARCHICAL DRIFT CONSTRAINTS FOR INCREMENTAL VISION-LANGUAGE OBJECT DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

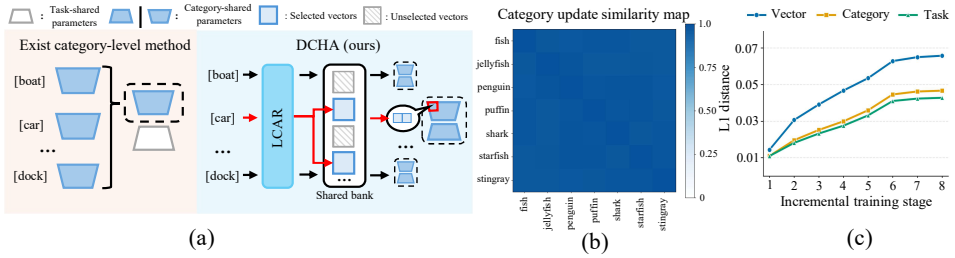
Incremental vision-language object detection (IVLOD) aims to continuously adapt vision-language detectors to emerging domains while preserving previously acquired knowledge and open-vocabulary generalization. While current methods typically optimize LoRA experts to achieve efficient adaptation, these training paradigms exhibit poor plasticity, as empirically evidenced by highly homogeneous parameter update directions across different categories. This rigidity stems from sharing restricted parameter spaces. Specifically, task-level LoRA methods force all categories to share unified experts, while existing category-level LoRA methods rigidly assign experts based solely on labels, completely ignoring intra-class feature variance and imposing a fixed routing capacity. To break this rigidity, we propose category-adaptive composition (CAC) equipped with learnable category-adaptive routing (LCAR). Inspired by Sparsegen, LCAR utilizes learnable keys and thresholds to dynamically select and assemble the most matched fine-grained parameter vectors into exclusive category LoRA experts. This demand-aware routing ensures that update directions are highly tailored to category-specific feature requirements, thereby significantly enhancing parameter plasticity and reducing training interference. Furthermore, because CAC introduces a hierarchical parameter structure where vectors compose category experts and category experts aggregate into task experts, parameter deviations inevitably accumulate across levels. To mitigate this severe error accumulation, we introduce a hierarchical drift constraint (HDC) that performs layer-wise regularization. Extensive experiments on ODinW-13 and ODinW-O demonstrate that our framework consistently outperforms state-of-the-art methods.

1 INTRODUCTION

Benefiting from large-scale pre-training Shao et al. (2019); Kamath et al. (2021); Plummer et al. (2015), vision-language object detection (VLOD) models have demonstrated strong zero-shot recognition capabilities in open-vocabulary settings Li et al. (2022); Liu et al. (2024); Cheng et al. (2024); Wang et al. (2024). Nevertheless, their performance often deteriorates substantially when transferred to specialized downstream domains. Incremental Vision-Language Object Detection (IVLOD) Deng et al. (2024); Cappellino et al. (2026); Wang et al. (2026) addresses this limitation by continuously adapting VLOD models to sequentially arriving tasks using parameter-efficient fine-tuning (PEFT) Wang et al. (2023); Cao et al. (2026), such as LoRA Hu et al. (2022), to acquire new knowledge through lightweight low-rank updates.

Despite these advancements, current LoRA-based IVLOD methods suffer from a critical flaw in model plasticity. We empirically observe that parameter update directions across different categories are highly homogeneous, as shown in Fig. 1(b). The fundamental cause is that current methods confine fine-tuning parameters to a rigid shared space. Specifically, task-level methods force all categories within a task to share a unified set of expert parameters. Existing category-level LoRA methods Cappellino et al. (2026) attempt to alleviate this limitation by combining a task-shared module with a category-specific module, as illustrated in the left part of Fig. 1(a). Nevertheless, their assignment paradigm remains rigidly tied to discrete class labels. They deterministically assign a fixed category expert to all samples within a class, completely ignoring inherent feature variances.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) (b) (c)

Figure 1: Motivation of DCHA. (a) Existing category-level methods rigidly assign fixed experts based on category, whereas DCHA dynamically selects and composes fine-grained parameter vectors according to category demands. (b) Category-specific updates produced by existing category-level methods exhibit uniformly high pairwise similarities within the same task. (c) Shared bank parameter deviations progressively accumulate at the vector, category, and task levels during incremental training, resulting in hierarchical drift.

Such a blind assignment fails to account for instances where samples of the same category are distant in the feature space, or samples of different categories are closely distributed. Furthermore, their expert selection paradigm is structurally inflexible in two critical aspects. First, it activates a fixed number of experts for every category. Because categories inherently possess varying levels of semantic complexity, this uniform capacity allocation fails to satisfy their heterogeneous demands. Second, some category-level LoRA methods Cappellino et al. (2026) rely on a unified routing network to sequentially learn diverse categories. As new categories are continuously introduced, the shared routing weights suffer from severe parameter overwriting, which directly triggers catastrophic forgetting within the routing mechanism itself.

To break this rigidity, we propose DCHA, a **D**ynamic **C**ompositional and **H**ierarchically Constrained **A**daptation framework for parameter-efficient IVLOD. DCHA is centered on category-adaptive composition (CAC) equipped with learnable category-adaptive routing (LCAR), as illustrated in the right part of Fig. 1(a). Instead of relying on a restricted parameter space, CAC decouples adaptation knowledge into a shared bank of fine-grained parameter vectors. Specifically, it dynamically assembles these vectors to construct exclusive category experts, which are subsequently aggregated to form task-level experts. Crucially, LCAR takes inspiration from the Sparsegen mechanism Laha et al. (2018) to provide a demand-aware selection strategy for these fine-grained parameter vectors. In LCAR, each category is assigned an exclusive learnable key vector and a specific threshold. Each element within this key vector explicitly corresponds to a distinct parameter vector in the shared bank, which embeds specific fine-grained features. During optimization, if an element's magnitude within the key vector exceeds the learned threshold, its corresponding parameter vector is activated and selected for composition. Since different categories have varying demands for diverse fine-grained features, this demand-aware routing provides an unconstrained and highly tailored expert selection mechanism. By assembling these dynamically selected vectors into exclusive category LoRA experts, the update directions align perfectly with the specific category demands. This strategy effectively enhances the diversity of parameter updates, and inherently preserves parameter plasticity.

However, the compositional nature of CAC inadvertently leads to hierarchical parameter drift, as shown in Fig. 1(c), posing a critical stability challenge for IVLOD. Because the architecture forms a progressive dependency where parameter vectors compose category experts that in turn aggregate into task-level experts, parameter deviations progressively accumulate across these hierarchical levels as the shared bank evolves. To address this structural side effect, we propose the hierarchical drift constraint (HDC). HDC performs layer-wise parameter regularization at the vector, category, and task levels, effectively cutting off the accumulation of representation drift and stabilizing the model. We evaluate DCHA on ODinW-13 Li et al. (2022) under the standard IVLOD setting and on ODinW-O. Experimental results demonstrate that DCHA consistently outperforms state-of-the-art methods across both settings. Our main contributions are summarized as follows:

- We identify the plasticity bottleneck in current IVLOD paradigms and propose CAC with LCAR. By utilizing dynamic, Sparsegen-inspired demand-aware routing, we construct highly tailored, exclusive LoRA experts for each category, significantly enhancing parameter plasticity.

- We propose HDC to perform layer-wise regularization across the vector, category, and task levels. This effectively cuts off the hierarchical accumulation of representation drift induced by compositional structures, thereby stabilizing the model.
- Extensive experiments on the ODinW-13 and ODinW-O benchmarks demonstrate that DCHA consistently outperforms state-of-the-art IVLOD methods, achieving a favorable balance between incremental adaptation and knowledge preservation.

2 Related Work

Vision-Language Object Detection (VLOD). VLOD also commonly studied as Open-Vocabulary Object Detection (OVOD), aims to detect objects specified by arbitrary category names, including categories unseen during training. Early approaches Gu et al. (2021); Zhong et al. (2022); Zang et al. (2022) transfer the image–text representations learned by CLIP Radford et al. (2021) to object detection, but require additional region-level alignment to bridge image-level pre-training and precise object localization. Recent developments have followed two main directions. The first incorporates object detection into Large Multimodal Models (LMMs) by formulating recognition and localization as autoregressive vision-language generation. Representative works in the Griffon series Zhan et al. (2024b; 2025; 2024a) progressively equip general-purpose LMMs Team et al. (2024); Touvron et al. (2023) with multi-object localization, high-resolution dense detection, and unified vision-centric and vision-language modeling. Nevertheless, these models incur substantial computational and parameter costs while still exhibiting limited open-vocabulary detection capabilities. The second direction develops specialized VLOD models through large-scale vision-language alignment pre-training, including GLIP Li et al. (2022), GroundingDINO Liu et al. (2024), YOLO-World Cheng et al. (2024), and OV-DINO Wang et al. (2024). These models reduce computational and parameter costs while enhancing open-vocabulary recognition and localization, but their performance still deteriorates when transferred to specialized downstream domains. Sequentially adapting them to new domains may further overwrite previously acquired knowledge and compromise the open-vocabulary generalization inherited from pre-training.

Incremental Vision-Language Object Detection (IVLOD). IVLOD extends VLOD to the continual learning setting, requiring models to adapt to sequentially introduced domains and categories while preserving both previously acquired knowledge and the zero-shot generalization capability. Traditional Incremental Object Detection (IOD) methods, such as CL-DETR Liu et al. (2023), TFA Wang et al. (2020), AT Houlsby et al. (2019), and iDETR Dong et al. (2023), are designed for closed-set continual detection without open-vocabulary recognition capabilities. OW-DETR Gupta et al. (2022) first introduces the open-world setting into IOD, but its generalization capability remains constrained by the closed-set model. To overcome this limitation, existing IVLOD methods build upon VLOD models and adopt Parameter-Efficient Fine-Tuning (PEFT) techniques, such as prompt tuning Wang et al. (2023); Cao et al. (2026), adapters Yu et al. (2024); He et al. (2021), and Low-Rank Adaptation (LoRA) Li et al. (2026; 2024), to enable efficient incremental adaptation while preserving open-vocabulary capabilities. ZiRa Deng et al. (2024) applies dual-norm regularization to the residual branches of the visual and textual encoders, and DGS Wang et al. (2026) constructs dynamic group-wise adapter subspaces for cross-task knowledge sharing. Most existing IVLOD methods operate at the task level. DitHub Cappellino et al. (2026) extends adaptation to the category level by combining a task-shared module with a category-specific module, enabling finer-grained adaptation for different categories. However, confining category updates to a task-shared adaptation subspace and assigning uniform capacity despite heterogeneous requirements jointly produce highly correlated and homogeneous category adaptations. Moreover, updating shared parameters causes historical category and task adaptations to drift, exacerbating forgetting. In this work, we propose a parameter-efficient IVLOD framework for effective category-level adaptation to downstream domains while preserving historical knowledge and open-vocabulary generalization.

3 Methodology

3.1 Problem Setting

We consider a sequence of incremental tasks $\mathcal{T} = \{D_1, D_2, \ldots, D_T\}$, where each task D_t contains images annotated with categories from $\mathcal{C}_t$. Specifically, each sample in D_t is represented as

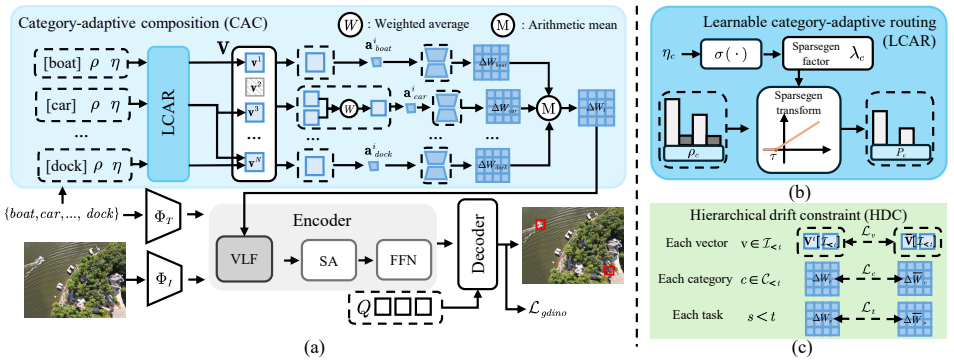


Figure 2: Overview of the proposed DCHA framework. (a) DCHA constructs category-specific LoRA by composing vectors from a shared bank and averages the resulting category updates to form the task-level adaptation for the VLF layers. (b) LCAR transforms category-specific routing logits into sparse composition weights using a learnable sparsity coefficient, enabling category-dependent vector selection. (c) HDC preserves historical adaptation knowledge by constraining deviations at the bank-vector, category-update, and task-update levels.

(x, y), where x denotes an input image and $y = \{(b_i, c_i)\}_{i=1}^{N}$ contains object annotations, including bounding boxes b_i and category labels $c_i \in \mathcal{C}_t$. Unlike conventional incremental detection settings, IVLOD allows overlapping categories across tasks (i.e., $\mathcal{C}_t \cap \mathcal{C}_{t'} \neq \emptyset$). The goal of IVLOD is to adapt the model to the current task D_t while retaining knowledge from previous tasks $\{D_1, \ldots, D_{t-1}\}$ and preserving zero-shot generalization capability on the general domain D_{ZS}.

3.2 Overview of DCHA

The overall pipeline of our proposed DCHA framework is illustrated in Fig. 2. DCHA builds upon the pre-trained GroundingDINO Liu et al. (2024), which comprises an image backbone Φ_I, a text backbone Φ_T, and a transformer-based encoder–decoder detector Φ_D. Given an input image x and the category set $\mathcal{C}_t$ of task t, the image backbone extracts visual features E_i, while the text backbone encodes the category names into text features E_t. The detector performs vision-language fusion over the visual features, text features, and object queries Q, producing object embeddings O and bounding boxes B. Detection scores C are obtained by measuring the similarity between the object and text embeddings. The overall forward process is formulated as

$$E_i = \Phi_I(x), \qquad E_t = \Phi_T(\mathcal{C}_t), \qquad \{O, B\} = \Phi_D(E_i, E_t, Q), \qquad C = OE_t^{\top}, \tag{1}$$

where $\top$ denotes the transpose operation. During incremental training, we freeze all pre-trained parameters and introduce Low-Rank Adaptation (LoRA) Hu et al. (2022) into the Vision–Language Fusion (VLF) layers in the detector. Given a frozen projection matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA parameterizes its update as $\Delta W = BA$, where $A \in \mathbb{R}^{r\times k}$, $B \in \mathbb{R}^{d\times r}$, and $r \ll \min(d, k)$. The adapted projection is therefore given by $W = W_0 + \Delta W$, with only the additional adaptation parameters being optimized. For each LoRA-injected layer, Category-Adaptive Composition (CAC) constructs both LoRA modules for each category by composing learnable vectors from a shared vector bank, yielding a category-specific update ΔW_c. Within CAC, Learnable Category-Adaptive Routing (LCAR) selects a category-dependent subset of bank vectors for composition, enabling distinctive adaptation patterns and effective capacities tailored to heterogeneous category requirements. Hierarchical drift constraint (HDC) further regulates the evolution of the shared vector bank and its induced adaptations at the bank-vector, category-update, and task-update levels. For task t, its category-specific updates are aggregated into a unified task-level update $\Delta W_t = \frac{1}{|\mathcal{C}_t|}\sum_{c\in\mathcal{C}_t} \Delta W_c$.

3.3 Category-Adaptive Composition

Existing category-level adaptation methods typically combine a task-shared LoRA module with independently learned category-specific LoRA. Consequently, all category updates remain confined to a common task-level adaptation subspace and may exhibit homogeneous adaptation patterns, restricting the expressiveness of category-specific adaptation. To address these limitations, we propose

Category-Adaptive Composition (CAC). For each LoRA-injected layer, we maintain a shared vector bank $\mathbf{V} = [\mathbf{v}^1, \ldots, \mathbf{v}^N]^\top \in \mathbb{R}^{N \times L}$, containing N adaptation vectors of length L. For each category $c \in \mathcal{C}_t$ and rank index $i \in [r]$, the corresponding LoRA rank module consists of two vectors, $\mathbf{a}_c^i \in \mathbb{R}^k$ and $\mathbf{b}_c^i \in \mathbb{R}^d$. We partition them into non-overlapping blocks of length L, yielding $U = k/L$ blocks for $\mathbf{a}_c^i$ and $Q = d/L$ blocks for $\mathbf{b}_c^i$. The j-th blocks of these vectors are denoted by $\mathbf{a}_c^{i,j} \in \mathbb{R}^L$ and $\mathbf{b}_c^{i,j} \in \mathbb{R}^L$, where $j \in [U]$ and $j \in [Q]$, respectively. For each block, category c maintains a routing logits $\boldsymbol{\rho}_c^{A,i,j} \in \mathbb{R}^N$ for $j \in [U]$ and $\boldsymbol{\rho}_c^{B,i,j} \in \mathbb{R}^N$ for $j \in [Q]$. Each routing logit scores the N candidate bank vectors. The corresponding composition weights $\mathbf{P}_c^{A,i,j}$ and $\mathbf{P}_c^{B,i,j}$ are obtained as

$$\mathbf{P}_c^{A,i,j} = \mathcal{S}\left(\boldsymbol{\rho}_c^{A,i,j}\right) \in \mathbb{R}^N, \qquad \mathbf{P}_c^{B,i,j} = \mathcal{S}\left(\boldsymbol{\rho}_c^{B,i,j}\right) \in \mathbb{R}^N, \tag{2}$$

where $\mathcal{S}$ denotes the routing transformation, as detailed in the Learnable Category-Adaptive Routing (LCAR) subsection. These weights retrieve and combine the corresponding bank vectors to construct each local block:

$$\mathbf{a}_c^{i,j} = \mathbf{V}^\top \mathbf{P}_c^{A,i,j} \in \mathbb{R}^L, \qquad \mathbf{b}_c^{i,j} = \mathbf{V}^\top \mathbf{P}_c^{B,i,j} \in \mathbb{R}^L. \tag{3}$$

The resulting local blocks are concatenated along the feature dimension to form the complete rank-wise vectors:

$$\mathbf{a}_c^i = \left[\left(\mathbf{a}_c^{i,1}\right)^\top \quad \cdots \quad \left(\mathbf{a}_c^{i,U}\right)^\top\right]^\top \in \mathbb{R}^k, \quad \mathbf{b}_c^i = \left[\left(\mathbf{b}_c^{i,1}\right)^\top \quad \cdots \quad \left(\mathbf{b}_c^{i,Q}\right)^\top\right]^\top \in \mathbb{R}^d. \tag{4}$$

Finally, the r rank-wise vectors are assembled into the complete category-specific LoRA modules:

$$A_c = \left[\mathbf{a}_c^1 \quad \cdots \quad \mathbf{a}_c^r\right]^\top \in \mathbb{R}^{r \times k}, \quad B_c = \left[\mathbf{b}_c^1 \quad \cdots \quad \mathbf{b}_c^r\right] \in \mathbb{R}^{d \times r}, \quad \Delta W_c = \gamma B_c A_c. \tag{5}$$

where γ is the learnable LoRA scaling coefficient. Instead of relying on a task-shared LoRA module, CAC constructs both LoRA modules for each category through block-wise composition over the shared vector bank. This hierarchical construction provides a more expressive parameterization for category-specific adaptation.

3.4 Learnable Category-Adaptive Routing

Existing category-level adaptation methods employ category-specific parameters in the same predefined manner for every category, without adapting the amount of adaptation resources to their individual requirements. In vector-based composition, fixed routing retains this limitation by activating a predefined number of bank vectors for all categories, thereby imposing uniform effective adaptation capacity. To accommodate such differences, we introduce Learnable Category-Adaptive Routing (LCAR) within CAC. We adopt Sparsegen Laha et al. (2018) because it produces differentiable routing weights with controllable support, allowing the routing structure to be optimized end-to-end. For each block-wise routing logits $\boldsymbol{\rho}_c^{X,i,j} \in \mathbb{R}^N$, where $X \in \{A, B\}$, LCAR applies Sparsegen Laha et al. (2018) by optimizing a candidate composition-weight vector $\mathbf{p}$ over the probability simplex:

$$\mathbf{P}_c^{X,i,j} = \underset{\mathbf{p} \in \Delta^{N-1}}{\arg\min} \left\{ \left\| \mathbf{p} - \boldsymbol{\rho}_c^{X,i,j} \right\|_2^2 - \lambda_c \left\| \mathbf{p} \right\|_2^2 \right\}, \qquad \Delta^{N-1} = \left\{ \mathbf{p} \in \mathbb{R}_+^N \,\middle|\, \mathbf{1}^\top \mathbf{p} = 1 \right\}. \tag{6}$$

Here, Δ^{N-1} denotes the probability simplex, $\mathbf{P}_c^{X,i,j}$ is the resulting optimal composition-weight vector, and $\lambda_c < 1$ is the Sparsegen sparsity coefficient. Rather than fixing this coefficient across all categories, we make it category-specific to adapt the number of activated bank vectors: $\lambda_c = (1 - \epsilon)\sigma(\eta_c)$, where η_c is a learnable scalar, $\sigma(\cdot)$ is the sigmoid function, and ϵ is a small constant. Equation equation 6 can be evaluated in closed form. Let $e_n = \rho_c^{X,i,j,n}$ denote the n-th routing logit, and sort the logits as $e_{(1)} \geq \cdots \geq e_{(N)}$. Defining $C_m = \sum_{k=1}^m e_{(k)}$, the active-set size $m^\star$ and corresponding threshold $\tau_c^{X,i,j}$ are

$$m^\star = \max \left\{ m \in [N] \,\middle|\, 1 - \lambda_c + m e_{(m)} > C_m \right\}, \qquad \tau_c^{X,i,j} = \frac{C_{m^\star} - 1 + \lambda_c}{m^\star}. \tag{7}$$

The resulting weight for the n-th bank vector is

$$P_c^{X,i,j,n} = \left[\frac{e_n - \tau_c^{X,i,j}}{1 - \lambda_c} \right]_+, \quad [x]_+ = \max(x, 0), \quad P_c^{X,i,j,n} = 0 \text{ if } e_n \leq \tau_c^{X,i,j}. \tag{8}$$

Consequently, LCAR provides category-adaptive control over bank-vector composition through the routing logits and the learnable sparsity coefficient. The routing logits determine the selection and relative contributions of bank vectors, while λ_c adaptively controls the number of activated vectors. Together, they enable each category to construct a tailored composition of bank vectors according to its specific adaptation requirements, thereby enhancing the flexibility and distinctiveness of category specialization.

Table 1: Comparison of mAP on ODinW-13 under different shot settings.

Shots	Method	ZCOCO	Avg	H	Ae	Aq	Co	Eg	Mu	Pa	Pv	Pi	Po	Ra	Sh	Th	Ve
0	G-DINO	47.4	47.0	47.2	19.1	20.8	64.8	60.0	25.3	56.3	54.8	65.9	22.1	62.0	32.9	70.4	57.1
5	TFA	26.4	45.8	33.5	21.9	22.3	67.4	60.7	30.6	53.6	46.8	63.6	26.9	56.3	28.0	64.3	52.5
	iDETR	43.5	51.7	47.2	25.7	25.5	70.4	63.0	50.0	50.5	54.9	64.8	**33.2**	57.6	42.4	**76.5**	56.9
	AT	43.7	47.2	45.4	14.6	25.0	66.6	**64.2**	38.9	42.0	55.5	**65.2**	27.5	52.7	32.2	71.2	57.5
	ZiRa	45.4	51.8	48.4	24.2	26.4	71.3	64.0	49.5	60.5	60.1	64.4	32.2	56.3	40.6	69.2	54.3
	DitHub	45.9	52.9	49.2	**26.8**	**28.9**	68.5	60.3	**51.9**	55.1	60.4	59.4	**33.2**	**68.1**	38.1	75.0	**61.3**
	DCHA (ours)	**46.0**	**53.5**	**49.5**	25.0	26.5	**71.9**	63.5	50.5	**68.3**	**61.5**	58.8	29.9	61.1	**42.5**	**76.5**	59.2
10	TFA	34.4	46.6	39.6	21.2	22.2	66.8	60.6	32.4	50.2	55.5	65.0	27.6	57.4	28.1	66.8	52.1
	iDETR	43.5	53.3	47.9	25.4	27.7	65.6	**67.6**	48.0	60.2	56.3	63.9	35.2	59.4	**53.6**	74.7	55.1
	AT	43.1	47.3	45.1	18.7	25.4	69.8	66.3	35.8	48.3	53.4	64.1	28.9	50.3	32.5	66.6	55.4
	ZiRa	46.1	53.2	49.4	25.6	26.0	71.7	65.3	48.2	**62.8**	61.2	65.5	35.6	55.7	46.9	71.0	56.1
	DitHub	46.5	54.4	50.1	26.2	**30.3**	68.1	65.7	**52.8**	56.8	**63.4**	61.9	**36.7**	**66.8**	41.5	76.0	**61.5**
	DCHA (ours)	**47.0**	**54.6**	**50.5**	**27.1**	28.6	**72.6**	65.5	49.8	59.2	63.2	**66.9**	31.4	63.9	45.7	**76.1**	59.6
Full	TFA	31.0	47.9	37.6	23.8	30.7	67.2	61.8	30.5	50.2	47.7	60.9	29.3	61.7	31.4	66.2	61.6
	iDETR	37.3	58.7	45.6	32.6	46.7	71.0	68.6	**55.3**	58.9	64.5	71.0	50.3	63.3	39.2	77.1	64.8
	AT	42.3	51.1	46.3	23.6	39.9	72.3	65.5	31.5	50.5	60.5	66.1	39.1	53.5	34.0	68.1	60.2
	OW-DETR	31.2	55.6	40.0	28.5	43.8	70.5	67.8	43.8	56.8	63.1	69.5	45.2	59.0	37.0	74.4	63.2
	CL-DETR	32.2	57.3	41.2	29.4	45.2	71.9	69.9	45.2	58.5	65.1	71.7	46.6	60.8	38.1	76.7	65.2
	ZiRa	46.3	58.0	51.5	31.8	47.4	71.8	64.7	46.5	62.7	66.4	71.0	48.5	63.0	41.4	76.1	62.4
	DitHub	47.0	62.2	53.5	34.6	50.7	70.5	68.6	49.3	**65.6**	69.6	71.1	**56.7**	70.9	52.8	79.3	68.2
	DGS	46.4	60.9	52.7	**40.3**	**54.8**	60.5	**78.3**	42.6	59.1	58.4	**75.9**	56.1	69.5	50.5	79.4	66.4
	DCHA (ours)	**47.8**	**62.3**	**54.1**	35.0	46.6	**73.9**	71.2	53.3	63.4	**70.5**	70.7	50.6	**72.1**	**53.0**	**80.8**	**68.5**

3.5 HIERARCHICAL DRIFT CONSTRAINT

As the shared vector bank is continuously updated to accommodate incoming tasks, changes in its vectors can alter the adaptations of previously learned categories and further propagate to their task-level representations. To mitigate such accumulated drift, we propose the Hierarchical Drift Constraint (HDC), which preserves historical adaptation knowledge at the vector, category, and task levels. Before learning task t, we keep the routing logits of all historical categories $c \in \mathcal{C}_{<t}$ fixed and store the bank after task $t-1$ as a snapshot $\bar{\mathbf{V}}$. We define the bank vectors activated by category c and by all historical categories as

$$\mathcal{I}_c = \left\{ n \in [N] \,\middle|\, \exists\, X \in \{A, B\},\, i,\, j : P_c^{X,i,j,n} > 0 \right\}, \qquad \mathcal{I}_{<t} = \bigcup_{c \in \mathcal{C}_{<t}} \mathcal{I}_c. \tag{9}$$

At the vector level, we constrain all bank vectors previously used by historical categories:

$$\mathcal{L}_v = \mathcal{D}\big(\mathbf{V}^t[\mathcal{I}_{<t}]\,, \bar{\mathbf{V}}[\mathcal{I}_{<t}]\big)\,, \tag{10}$$

where $\mathbf{V}^t$ is the bank being optimized for task t and $\mathcal{D}$ denotes the Smooth-L_1 distance. At the category level, the fixed historical routing weights reconstruct the category updates $\Delta\mathbf{W}_c^t$ and $\Delta\overline{\mathbf{W}}_c$ from $\mathbf{V}^t$ and $\bar{\mathbf{V}}$, respectively. We then introduce $\mathcal{L}_c$ to constrain the discrepancy between these reconstructed updates:

$$\mathcal{L}_c = \frac{1}{|\mathcal{C}_{<t}|} \sum_{c \in \mathcal{C}_{<t}} \mathcal{D}\big(\Delta W_c^t, \Delta\overline{W}_c\big)\,. \tag{11}$$

At the task level, we aggregate these category updates within each historical task $s < t$ and constrain their overall drift:

$$\Delta W_s = \frac{1}{|\mathcal{C}_s|} \sum_{c \in \mathcal{C}_s} \Delta W_c, \quad \Delta\overline{W}_s = \frac{1}{|\mathcal{C}_s|} \sum_{c \in \mathcal{C}_s} \Delta\overline{W}_c, \quad \mathcal{L}_t = \frac{1}{t-1} \sum_{s=1}^{t-1} \mathcal{D}\big(\Delta W_s, \Delta\overline{W}_s\big)\,. \tag{12}$$

$\mathcal{L}_v$ preserves historical bank vectors activated by the current task while leaving the remaining bank capacity available for new adaptations. $\mathcal{L}_c$ prevents bank-vector drift from accumulating into distorted category updates, whereas $\mathcal{L}_t$ maintains the overall adaptation structure of previous tasks beyond individual categories. Finally, we combine the three HDC terms with the standard GroundingDINO detection loss $\mathcal{L}_{\text{gdino}}$. The overall training objective is

$$\mathcal{L} = \mathcal{L}_{\text{gdino}} + \beta_v \mathcal{L}_v + \beta_w \mathcal{L}_c + \beta_t \mathcal{L}_t, \tag{13}$$

where β_v, β_c, and β_t are the coefficients for balancing the three levels of regularization.

Table 2: Comparison of mAP on ODinW-O.

Method	ZCOCO	Avg	H	Ae	Hw	Pv	Sd	Th	Ve
G-DINO	47.4	53.2	50.1	45.1	67.5	58.1	25.8	70.4	51.9
iDETR	37.4	59.0	45.8	**49.7**	67.8	67.7	36.0	**78.4**	54.6
AT	44.9	56.9	50.2	46.9	65.8	67.3	30.6	76.0	54.8
ZiRa	44.4	57.6	50.1	39.9	**68.0**	64.9	**46.3**	77.3	49.5
DitHub	47.2	59.2	52.5	49.2	66.9	72.9	38.7	74.7	52.9
DCHA (ours)	**47.6**	**60.2**	**53.1**	45.8	67.8	**73.0**	42.0	75.7	**56.6**

Table 3: Comparison of catastrophic forgetting.

Method	ODinW-13		ODinW-O	
	$\Delta_{\mathrm{ZS}}\downarrow$	$\bar{F}\downarrow$	$\Delta_{\mathrm{ZS}}\downarrow$	$\bar{F}\downarrow$
ZiRa	1.1	4.8	3.0	4.3
DitHub	0.4	2.8	0.2	6.1
DCHA (ours)	**-0.4**	**0**	**-0.2**	**3.0**

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate our method on the ODinW-13 benchmark, which contains 13 diverse subtasks covering various domains and object categories. To evaluate category recurrence and module reuse, we further introduce ODinW-O, a subset of ODinW35 with higher category overlap, where most tasks contain categories observed in previous tasks. Following the IVLOD setting, all tasks are trained sequentially and evaluated after completing the entire sequence. The retained zero-shot generalization capability is measured on COCO Lin et al. (2014). By default, ablation studies are conducted on ODinW-13, with further details provided in the appendix.

Baselines. We compare DCHA with representative methods from both IOD and IVLOD. For IOD baselines, CL-DETR Liu et al. (2023) and OW-DETR Gupta et al. (2022) are DETR-based continual detection methods, while TFA Wang et al. (2020), AT Houlsby et al. (2019), and iDETR Dong et al. (2023) investigate few-shot and parameter-efficient adaptation. For IVLOD baselines, we compare with ZiRa Deng et al. (2024), DitHub Cappellino et al. (2026), and DGS Wang et al. (2026), which introduce lightweight adaptation strategies.

Metrics. We report the average mAP Dave et al. (2021) after sequentially training on all tasks, denoted as Avg, where mAP is calculated as the mean $\mathrm{AP}_{50\text{–}95}$ over all categories. To evaluate the trade-off between continual adaptation and preservation of open-vocabulary generalization, we adopt the harmonic mean Xian et al. (2017):

$$\mathrm{H} = \frac{2 \times (\mathrm{ZCOCO} \times \mathrm{Avg})}{\mathrm{ZCOCO} + \mathrm{Avg}}, \tag{14}$$

where ZCOCO denotes the mAP on zero-shot COCO. To measure forgetting, we further introduce zero-shot forgetting Δ_{ZS} and average task forgetting $\bar{F}$ Dave et al. (2021). Specifically, Δ_{ZS} denotes the degradation in COCO zero-shot performance after learning all tasks. $\bar{F}$ is defined as

$$\bar{F} = \frac{1}{T-1} \sum_{i=1}^{T-1} \left(\max_{t \in \{i,\ldots,T-1\}} R_{t,i} - R_{T,i} \right), \tag{15}$$

where $R_{t,i}$ denotes the mAP on task i after learning task t. Lower Δ_{ZS} and $\bar{F}$ indicate less forgetting.

Implementation Details. We adopt GroundingDINO-T (G-DINO) Liu et al. (2024) as the base detector, initialized with weights pre-trained on Objects365 Shao et al. (2019), GoldG Li et al. (2022), and Cap4M Li et al. (2022). For our method, LoRA modules are inserted only into the vision-language fusion (VLF) layers. During training, all pre-trained parameters are frozen, and only LoRA parameters with rank $r = 16$ are optimized. We use AdamW Loshchilov & Hutter with an initial learning rate of 2×10^{-3} and weight decay of 10^{-2}. The shared bank size and block length are set to $N = 40$ and $L = 256$. The HDC loss weights are set to $\beta_v = 0.7$, $\beta_w = 0.5$, and $\beta_t = 0.7$. Further details are provided in the appendix.

4.2 Main Results

Performance on ODinW-13. As shown in Table 1, our proposed DCHA achieves the highest ZCOCO, Avg, and H under the 5-shot, 10-shot, and full-data settings. In the full-data setting, DCHA substantially outperforms conventional IOD methods such as iDETR Dong et al. (2023) and CL-DETR Liu et al. (2023). Specifically, compared with iDETR, DCHA improves ZCOCO, Avg, and H by 10.5, 3.6, and 8.5 points, respectively. This stark contrast highlights that conventional incremental learning techniques are inadequate for balancing downstream task adaptation with the preservation of pre-trained open-vocabulary knowledge. Among existing IVLOD methods, DCHA

also achieves the best overall performance, reaching 47.8 ZCOCO, 62.3 Avg, and 54.1 H. DCHA remains effective under data-scarce settings. Compared with the second-best method, DitHub Cappellino et al. (2026), DCHA improves ZCOCO, Avg, and H by 0.1, 0.6, and 0.3 points in the 5-shot setting, and by 0.5, 0.2, and 0.4 points in the 10-shot setting, respectively. These results support the effectiveness of DCHA in adapting to downstream domains while mitigating forgetting of open-vocabulary knowledge.

Performance on ODinW-O. To further evaluate DCHA under stronger category overlap across incremental tasks, we conduct experiments on ODinW-O. As shown in Table 2, DCHA achieves the highest ZCOCO, Avg, and H, reaching 47.6, 60.2, and 53.1, respectively. Compared with DitHub Cappellino et al. (2026), these results represent improvements of 0.4, 1.0, and 0.6 points. Although other methods lead on individual datasets, DCHA achieves the best overall performance, demonstrating its effectiveness in the presence of recurring categories.

Forgetting on ODinW-13 and ODinW-O. Furthermore, Table 3 shows that DCHA consistently achieves the lowest forgetting on both benchmarks. On ODinW-13, DCHA obtains $\Delta_{\mathrm{ZS}} = -0.4$ and $\bar{F} = 0$, compared with 0.4 and 2.8 for DitHub. On ODinW-O, where categories recur more frequently across tasks, DCHA maintains a negative Δ_{ZS} of -0.2 and reduces average forgetting from DitHub's 6.1 to 3.0, an improvement of 3.1 points. These results show that DCHA preserves zero-shot capability and historical task performance, thereby mitigating catastrophic forgetting.

4.3 Model Analysis and Ablation Studies

Effectiveness of different components. We progressively ablate the three key components of DCHA: CAC, LCAR, and HDC. As shown in Table 4, the baseline learns an independent pair of LoRA modules for each category. CAC reduces the number of adaptation parameters from 123.8M to 20.7M, but fixed routing alone lowers Avg and H. Adding LCAR improves Avg by 3.7 points and H by 1.4 points, showing the benefit of adapting vector composition to each category. Its average forgetting also rises from 5.3 to 6.1, however. Introducing HDC raises Avg to 62.3 and H to 54.1 while reducing average forgetting to 0. Compared with the baseline, the complete DCHA improves Avg by 5.6 points and H by 2.0 points using only 20.8M adaptation parameters, while mitigating catastrophic forgetting.

Table 4: Component ablation of DCHA.

CAC	LCAR	HDC	Params	ZCOCO	Avg	H	$\bar{F}\downarrow$
			123.8M	**48.2**	56.7	52.1	4.5
✓			20.7M	47.5	52.0	49.6	5.3
✓	✓		20.8M	47.0	55.7	51.0	6.1
✓	✓	✓	20.8M	47.8	**62.3**	**54.1**	**0**

Shared bank size N. We investigate the effect of shared bank size in Table 5. Increasing N from 10 to 40 raises Avg from 58.8 to 62.3 and H from 52.7 to 54.1, whereas both metrics decline when N increases further. A larger bank provides more vectors for category-specific composition, but additional vectors may introduce redundancy without improving adaptation. At $N = 40$, DCHA achieves the highest Avg and H using 20.8M parameters. We therefore set $N = 40$.

Table 5: Effect of the shared bank size N.

N	Params	ZCOCO	Avg	H
10	5.2M	47.7	58.8	52.7
20	10.4M	**47.9**	60.2	53.4
40	20.8M	47.8	**62.3**	**54.1**
60	31.1M	47.6	61.4	53.6
80	41.5M	47.4	61.2	53.4

Block Size L. We examine the effect of block size in Table 6. Increasing L from 16 to 256 reduces the parameter count from 309.5M to 20.8M, while H remains between 53.8 and 54.2. Smaller blocks allow finer-grained composition but require substantially more parameters; larger blocks reduce this cost while retaining comparable performance. At $L = 256$, DCHA achieves 62.3 Avg and 54.1 H, only 0.6 and 0.1 points below the respective best results. We therefore set $L = 256$ to balance performance and parameter efficiency.

Table 6: Effect of the vector block size L.

L	Params	ZCOCO	Avg	H
16	309.5M	47.5	**62.9**	54.1
32	154.9M	47.6	62.8	**54.2**
64	77.7M	**47.8**	61.4	53.8
128	39.4M	**47.8**	61.8	53.9
256	**20.8M**	**47.8**	62.3	54.1

Effect of Routing Strategy in LCAR. We compare five routing strategies in Table 7. Top-1 and Top-2 use category-specific logits to select the one or two highest-scoring bank vectors, respectively. $\mathcal{S}$-MLP uses logits shared across categories and predicts the Sparsegen sparsity coefficient λ with an MLP, following LD-MOLE Zhuang et al. (2026). $\mathcal{S}$-Fixed uses category-specific logits with a fixed λ,

Table 7: Comparison of the routing strategies.

Strategy	ZCOCO	Avg	H
Top-1	47.8	60.3	53.3
Top-2	47.7	59.1	52.8
$\mathcal{S}$-MLP	44.3	59.5	50.8
$\mathcal{S}$-Fixed	**47.8**	60.6	53.4
LCAR (ours)	**47.8**	**62.3**	**54.1**

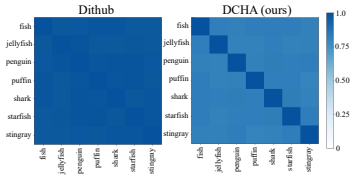


Figure 3: Pairwise CKA similarities among category-specific updates on Aq.

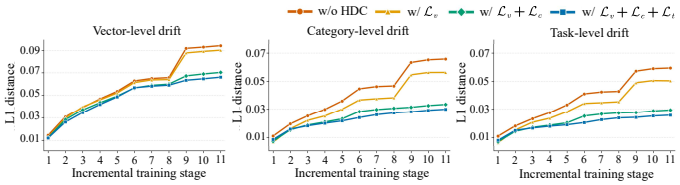


Figure 4: Vector-, category-, and task-level L1 distances from their states after the first task across subsequent incremental training stages under different HDC constraints.

while LCAR uses category-specific logits with an adaptive λ. Top-1 outperforms Top-2 in Avg and H (60.3/53.3 versus 59.1/52.8), and $\mathcal{S}$-MLP obtains 59.5 Avg and 50.8 H. Among these alternatives, $\mathcal{S}$-Fixed performs best, reaching 60.6 Avg and 53.4 H. These results suggest that selecting more vectors can introduce less relevant directions, while shared logits limit the ability to distinguish category-specific requirements. LCAR further improves Avg and H to 62.3 and 54.1, respectively, while matching the highest ZCOCO of 47.8. These gains demonstrate the effectiveness of our category-adaptive routing strategy for category-specific vector composition.

Effect of Hierarchical Drift Constraints. We progressively add the vector-, category-, and task-level constraints of HDC in Table 8. Without these constraints, the model obtains 51.0 H, 0.4 Δ_{ZS}, and 6.1 average forgetting. Adding the vector-level constraint raises H to 52.5 and reduces forgetting to 4.1. The category-level constraint further raises H to 53.5 and reduces forgetting to 1.1. With the task-level constraint, H reaches 54.1 and forgetting falls to 0; Δ_{ZS} also decreases progressively to -0.4. These gains suggest that stabilizing shared vectors alone is insufficient, and that constraining category- and task-level adaptations provides further protection. Together, the three constraints improve overall performance while mitigating catastrophic forgetting.

Table 8: Ablation study of the constraints in HDC.

$\mathcal{L}_v$	$\mathcal{L}_c$	$\mathcal{L}_t$	H	$\Delta_{\text{ZS}}\downarrow$	$\bar{F}\downarrow$
			51.0	0.4	6.1
✓			52.5	-0.1	4.1
✓	✓		53.5	-0.3	1.1
✓	✓	✓	**54.1**	**-0.4**	**0**

Visualization of Category Update Diversity. We visualize the pairwise CKA similarities among category-specific updates on Aq in Fig. 3 (the metric is defined in Appendix A.6.1). DitHub exhibits uniformly high off-diagonal similarities, indicating that its category updates are highly alike. In contrast, DCHA shows lower and more varied similarities across category pairs, suggesting that category-adaptive vector selection and composition produce more distinctive updates.

Visualization of Hierarchical Drift. We further visualize vector-, category-, and task-level drift across incremental stages in Fig. 4. Each stage denotes the model state after training on the corresponding task in the sequence, with drift measured relative to the representations after the first task. Without HDC, drift increases at all three levels, particularly in later stages. Applying $\mathcal{L}_v$ alone provides limited suppression because it does not directly constrain the composed category and task updates. Adding $\mathcal{L}_c$ substantially reduces category- and task-level drift, while the complete HDC with $\mathcal{L}_v + \mathcal{L}_c + \mathcal{L}_t$ produces the lowest and smoothest trajectories across all three levels. These results demonstrate the effectiveness of HDC in controlling representation drift during incremental learning. (the metric is defined in Appendix A.6.2)

5 Conclusion.

In this work, we propose DCHA, a parameter-efficient framework for IVLOD that addresses the tension between category-level plasticity and knowledge preservation. DCHA introduces CAC to represent category-specific LoRA adaptations through dynamic composition over a shared vector bank, while LCAR enables category-dependent routing and composition capacity. To mitigate the drift induced by continual evolution of shared adaptation parameters, HDC constrains the adaptation structure hierarchically at the vector, category, and task levels. Extensive experiments on ODinW-13 and ODinW-O demonstrate that DCHA achieves an effective balance between incremental adaptability and historical knowledge preservation, validating the effectiveness of our overall design.

REFERENCES

Weihao Cao, Runqi Wang, Xiaoyue Duan, Jinchao Zhang, Ang Yang, and Liping Jing. Parameter-efficient semantic augmentation for enhancing open-vocabulary object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20367–20376, 2026.

Chiara Cappellino, Gianluca Mancusi, Matteo Mosconi, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Dithub: A modular framework for incremental open-vocabulary object detection. *Adv. Neural Inform. Process. Syst.*, 38:22898–22926, 2026.

Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16901–16911, 2024.

Achal Dave, Piotr Dollár, Deva Ramanan, Alexander Kirillov, and Ross Girshick. Evaluating large-vocabulary object detectors: The devil is in the details. *arXiv preprint*, 2021.

Jieren Deng, Haojian Zhang, Kun Ding, Jianhua Hu, Xingxuan Zhang, and Yunkuan Wang. Zero-shot generalizable incremental learning for vision-language object detection. *Adv. Neural Inform. Process. Syst.*, 37:136679–136700, 2024.

Na Dong, Yongqiang Zhang, Mingli Ding, and Gim Hee Lee. Incremental-detr: Incremental few-shot object detection via self-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 543–551, 2023.

Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. *arXiv preprint*, 2021.

Akshita Gupta, Sanath Narayan, KJ Joseph, Salman Khan, Fahad Shahbaz Khan, and Mubarak Shah. Ow-detr: Open-world detection transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9235–9244, 2022.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint*, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *Int. Conf. Learn. Represent.*, 2022.

Aishwarya Kamath, Mannat Singh, Yann LeCun, Gabriel Synnaeve, Ishan Misra, and Nicolas Carion. Mdetr-modulated detection for end-to-end multi-modal understanding. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1780–1790, 2021.

Anirban Laha, Saneem Ahmed Chemmengath, Priyanka Agrawal, Mitesh Khapra, Karthik Sankaranarayanan, and Harish G Ramaswamy. On controllable sparse alternatives to softmax. *Adv. Neural Inform. Process. Syst.*, 31, 2018.

Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10965–10975, 2022.

Shiwei Li, Xiandi Luo, Haozhao Wang, Xing Tang, Ziqiang Cui, Dugang Liu, Yuhua Li, Xiuqiang He, and Ruixuan Li. Beyond higher rank: Token-wise input-output projections for efficient low-rank adaptation. *Adv. Neural Inform. Process. Syst.*, 38:136414–136435, 2026.

Yang Li, Shaobo Han, and Shihao Ji. Vb-lora: Extreme parameter efficient fine-tuning with vector banks. *Adv. Neural Inform. Process. Syst.*, 37:16724–16751, 2024.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European conference on computer vision*, pp. 38–55. Springer, 2024.

Yaoyao Liu, Bernt Schiele, Andrea Vedaldi, and Christian Rupprecht. Continual detection transformer for incremental object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23799–23808, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Int. Conf. Learn. Represent.*

Bryan A Plummer, Liwei Wang, Chris M Cervantes, Juan C Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2641–2649, 2015.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR, 2021.

Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 8430–8439, 2019.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint*, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint*, 2023.

Hao Wang, Pengzhen Ren, Zequn Jie, Xiao Dong, Chengjian Feng, Yinlong Qian, Lin Ma, Dongmei Jiang, Yaowei Wang, Xiangyuan Lan, et al. Ov-dino: Unified open-vocabulary detection with language-aware selective fusion. *arXiv preprint*, 2024.

Runqi Wang, Xiaoyue Duan, Guoliang Kang, Jianzhuang Liu, Shaohui Lin, Songcen Xu, Jinhu Lü, and Baochang Zhang. Attriclip: A non-incremental learner for incremental knowledge learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3654–3663, 2023.

Xin Wang, Thomas E Huang, Trevor Darrell, Joseph E Gonzalez, and Fisher Yu. Frustratingly simple few-shot object detection. In *International Conference on Machine Learning*, pp. 9919–9928. PMLR, 2020.

Xu Wang, Zihan Lin, Yixin Zhang, and Zilei Wang. Boosting vision-language models towards cross-domain incremental object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6249–6260, 2026.

Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning-the good, the bad and the ugly. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4582–4591, 2017.

Jiazuo Yu, Yunzhi Zhuge, Lu Zhang, Ping Hu, Dong Wang, Huchuan Lu, and You He. Boosting continual learning of vision-language models via mixture-of-experts adapters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23219–23230, 2024.

Yuhang Zang, Wei Li, Kaiyang Zhou, Chen Huang, and Chen Change Loy. Open-vocabulary detr with conditional matching. In *European conference on computer vision*, pp. 106–122. Springer, 2022.

Yufei Zhan, Hongyin Zhao, Yousong Zhu, Fan Yang, Ming Tang, and Jinqiao Wang. Griffon-g: Bridging vision-language and vision-centric tasks via large multimodal models. *arXiv preprint*, 2024a.

Yufei Zhan, Yousong Zhu, Zhiyang Chen, Fan Yang, Ming Tang, and Jinqiao Wang. Griffon: Spelling out all object locations at any granularity with large language models. In *European conference on computer vision*, pp. 405–422. Springer, 2024b.

Yufei Zhan, Shurong Zheng, Yousong Zhu, Hongyin Zhao, Fan Yang, Ming Tang, and Jinqiao Wang. Griffon v2: Advancing multimodal perception with high-resolution scaling and visual-language co-referring. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 22947–22957. IEEE, 2025.

Yiwu Zhong, Jianwei Yang, Pengchuan Zhang, Chunyuan Li, Noel Codella, Liunian Harold Li, Luowei Zhou, Xiyang Dai, Lu Yuan, Yin Li, et al. Regionclip: Region-based language-image pretraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16793–16803, 2022.

Yuan Zhuang, Yi Shen, Yuexin Bian, Qing Su, Shihao Ji, Yuanyuan Shi, and Fei Miao. Ld-mole: Learnable dynamic routing for mixture of lora experts. In *International Conference on Learning Representations*, volume 2026, pp. 64840–64858, 2026.

A Appendix

A.1 AI use statement

AI tools were used solely to assist with language editing and polishing during the preparation of this manuscript. All technical content, methodology, experiments, analyses, and conclusions were developed and verified by the authors.

A.2 Additional Experimental Implementation

For ODinW-13, we follow the training protocol of ZiRa Deng et al. (2024), while for ODinW-O, we follow DitHub Cappellino et al. (2026). All models are trained with a batch size of 4. All experiments use a fixed random seed of 42 and are conducted on a single NVIDIA L20 GPU. GPU memory consumption ranges from 5GB to 43GB depending on the training stage and task complexity, and the complete training process takes approximately 36 hours. Our implementation is based on Python 3.9.25, PyTorch 2.4.0 with CUDA 12.4, and Detectron2 0.6.

A.3 Additional Benchmark Details

ODinW-13 benchmark. Following ZiRa Deng et al. (2024), we evaluate sequential specialization on the Object Detection in the Wild benchmark Li et al. (2022) and assess the preservation of zero-shot generalization on COCO Lin et al. (2014). ODinW-13 consists of 13 heterogeneous object detection datasets: Aerial Maritime Drone (Ae), Aquarium (Aq), Cottontail Rabbits (Co), EgoHands (Eg), Mushrooms (Mu), Packages (Pa), PascalVOC (Pv), Pistols (Pi), Pothole (Po), Raccoon (Ra), Shellfish (Sh), Thermal(Th), and Vehicles (Ve). These datasets cover diverse object categories, visual environments and imaging conditions, providing a challenging setting for evaluating sequential adaptation across heterogeneous domains. To further evaluate open-vocabulary generalization, we use ZCOCO to denote zero-shot performance on the COCO validation set, a widely adopted benchmark in VLOD and OVOD that is also referred to as OV-COCO. The validation set contains 5,000 images annotated with 80 common object categories, such as person, car, and dog. After sequentially learning all ODinW-13 tasks, the model is evaluated on each learned task and MS COCO without additional fine-tuning. We use Avg to denote the average mAP across all learned tasks. Their harmonic mean H measures the balance between continual adaptation and open-vocabulary generalization.

ODinW-O benchmark. Following DitHub Cappellino et al. (2026), we additionally evaluate our method on ODinW-O, an overlapping-category benchmark derived from ODinW-35 Li et al. (2022). ODinW-O is designed to evaluate incremental adaptation when semantic categories recur across tasks under different domains. For example, the category car appears in aerial, street-level, and autonomous-driving scenes. The benchmark retains only categories shared by at least two tasks and removes tasks without recurring categories. The datasets and their overlapping categories in ODinW-O are as follows:

- **Aerial Maritime Drone**: boat and car;
- **Pascal VOC**: boat, car, dog, and person;
- **Thermal**: dog and person;
- **Vehicles**: car and truck;
- **Hard Hat Workers**: person;
- **Self-Driving Cars**: car and truck.

Apart from the task composition, all evaluation metrics follow the same definitions as those used for ODinW-13.

A.4 Normalization of Sparsegen Composition Weights

We show that the Sparsegen composition weights used in DSR are non-negative and sum to one. For clarity, we omit the category, block, rank, and LoRA-module indices and denote a routing vector by

Table 9: Parameter efficiency comparison. $\mathcal{E}$ and $\mathcal{F}$ denote encoder and vision-language fusion adaptation, respectively.

Method	Params	ZCOCO	Avg	H
G-DINO$^{\mathcal{E}}$	92M	38.6	54.7	45.3
G-DINO$^{\mathcal{F}}$	36M	46.1	57.0	51.0
ZiRa	18M	46.3	58.0	51.5
DitHub ($r = 16$)	147M	47.0	62.2	53.5
DCHA ($r = 1$)	2.6M	48.0	59.5	53.1
DCHA ($r = 2$)	3.8M	48.0	59.3	53.1
DCHA ($r = 4$)	6.3M	**48.2**	58.9	53.0
DCHA ($r = 8$)	11.1M	47.7	60.2	53.2
DCHA ($r = 16$)	20.8M	47.8	**62.3**	**54.1**
DCHA ($r = 32$)	40.1M	47.8	61.4	53.8

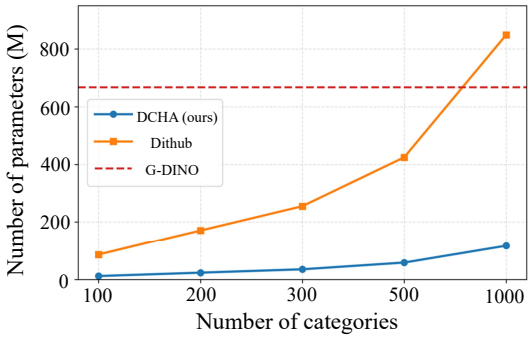


Figure 5: Parameter growth with the number of categories.

$\mathbf{R} \in \mathbb{R}^N$. Let $R^{(j)}$ denote its j-th largest entry and define $Z^m = \sum_{j=1}^{m} R^{(j)}$. Given the support size m^*, the corresponding threshold is $\tau = \frac{Z^{m^*} - 1 + \lambda}{m^*}$. The first m^* sorted entries receive positive weights, while the remaining entries are mapped to zero. Therefore,

$$\begin{aligned} \sum_{n=1}^{N} P^n &= \sum_{j=1}^{m^*} \frac{R^{(j)} - \tau}{1-\lambda} = \frac{Z^{m^*} - m^*\tau}{1-\lambda} \\ &= \frac{Z^{m^*} - \left(Z^{m^*} - 1 + \lambda\right)}{1-\lambda} = \frac{1-\lambda}{1-\lambda} = 1. \end{aligned} \tag{16}$$

Moreover, the positive-part operator guarantees $P^n \geq 0$ for every n. Hence, the resulting composition weights lie on the probability simplex:

$$\mathbf{P} \in \left\{ \mathbf{p} \in \mathbb{R}^N \mid p^n \geq 0, \sum_{n=1}^{N} p^n = 1 \right\}. \tag{17}$$

Consequently, each local LoRA block is constructed as a normalized convex combination of the selected bank vectors, while its overall adaptation magnitude is independently controlled by the category-dependent scaling coefficient.

A.5 Additional Experiments

Parameter Efficiency and Scalability. Table 9 compares DCHA at different LoRA ranks with existing methods. At $r = 16$, DCHA achieves the highest Avg (62.3) and H (54.1) using 20.8M parameters. Compared with DitHub, it uses 85.9% fewer parameters while improving H by 0.6 points. Even at $r = 1$, DCHA reaches 59.5 Avg and 53.1 H with only 2.6M parameters, exceeding ZiRa by 1.5 and 1.6 points, respectively. Increasing the rank to 32 nearly doubles the parameters to 40.1M, while H decreases to 53.8. Therefore, we choose $r = 16$. Figure 5 compares DCHA and DitHub at the same LoRA rank of $r = 16$. As the number of categories increases, DCHA's parameter count grows much more slowly because it reuses a shared vector bank and adds only lightweight category-specific routing parameters, enabling parameter-efficient expansion to larger category sets.

Effect of Vector-Bank Construction. Table 10 compares vector-bank construction strategies. Bank Scope indicates whether a bank is shared across categories within each layer or assigned to each category. A/B Banks indicates whether the two LoRAs use the same bank or separate banks. Our layer-wise shared-bank design achieves the highest ZCOCO, Avg, and H with the fewest parameters. Separating the A/B banks at the layer level increases parameters from 20.8M to 22.2M, while reducing

Table 10: Ablation study of different vector-bank construction strategies.

Bank Scope	A/B Banks	Params	ZCOCO	Avg	H
Layer-wise	Shared (ours)	**20.8M**	**47.8**	**62.3**	**54.1**
Layer-wise	Separate	22.2M	47.4	60.3	53.1
Category-wise	Shared	81.2M	47.1	61.4	53.3
Category-wise	Separate	143.1M	47.0	61.8	53.4

Avg by 2.0 points and H by 1.0 point. Category-wise banks require 81.2M or 143.1M parameters, depending on whether A/B share a bank, yet neither setting matches our performance. These results support reusing vectors across categories and LoRAs while constructing category-specific updates through routing.

Effect of Sparsity-Coefficient Granularity. We compare different granularities for learning the sparsity coefficient in Table 11. DCHA-Double learns separate coefficients for the A/B of LoRA, while DCHA-Block assigns an independent coefficient to each local block. DCHA instead shares one category-specific coefficient across both (A/B) and their blocks. Compared with these variants, DCHA improves Avg by 1.2 and 1.0 points, respectively, increases H by 0.5 points, and reduces average forgetting from 0.3 to 0. A shared coefficient does not impose identical activation patterns: independent block-wise routing logits still allow different blocks to select different numbers and combinations of vectors. Moreover, the coefficient and the logit scale have partially overlapping effects on routing concentration. Introducing separate coefficients therefore adds optimization freedom without necessarily improving effective expressiveness. The results suggest that category-level sharing provides a useful structural constraint, coordinating activation tendencies across blocks while preserving local flexibility through independent routing logits.

Table 11: Comparison of DCHA variants.

Method	ZCOCO↑	Avg↑	H↑	$\bar{F}\downarrow$
DCHA	**47.8**	**62.3**	**54.1**	**0**
DCHA-Double	47.7	61.1	53.6	0.3
DCHA-Block	47.6	61.3	53.6	0.3

Effect of Sparsity-Coefficient Parameterization. We compare four parameterizations of λ_c in Table 12. Direct parameterization clips the learnable scalar at the upper boundary, while the other variants use nonlinear mappings to satisfy $\lambda_c < 1$. Sigmoid achieves the best performance, reaching 47.8 ZCOCO, 62.3 Avg, and 54.1 H. Compared with the strongest alternative, exponential parameterization, it improves these metrics by 0.2, 1.2, and 0.6 points, respectively. It also surpasses direct and softplus parameterizations by 2.2 and 1.8 points in Avg. These results support the sigmoid parameterization for learning category-specific routing coefficients in LCAR.

Table 12: Ablation study of different parameterizations of λ_c in LCAR.

Parameterization	Mapping Function	ZCOCO	Avg	H
Direct	$\min(\eta_c,\, 1-\epsilon)$	47.7	60.1	53.2
Exponential	$1-\epsilon-\exp(\eta_c)$	47.6	61.1	53.5
Softplus	$1-\epsilon-\mathrm{softplus}(\eta_c)$	47.7	60.5	53.3
Sigmoid (ours)	$(1-\epsilon)\sigma(\eta_c)$	**47.8**	**62.3**	**54.1**

Comparison of routing balance. For each LoRA-injected layer, we compute the three routing-balance metrics from that layer's vector-selection counts and then average each metric across layers. CV measures the overall variation in how frequently different vectors are selected, where a lower value indicates more balanced usage. Entropy measures how evenly the selections are distributed across the entire vector bank, where a higher value indicates a more uniform distribution. Top-5 share denotes the proportion of selections assigned to the five most frequently used vectors, where a lower value indicates less selection concentration. Detailed computations are provided in Appendix A.7. As shown in Table 13, Fixed exhibits the strongest routing concentration, with the highest CV 0.72, the lowest entropy 0.93, and the largest Top-5 share 52.5%. Making λ directly learnable substantially improves all three metrics, demonstrating the importance of adapting the routing sparsity to different categories and layers. Softplus and Exponential further introduce positive reparameterizations of the learnable λ. Although they outperform Fixed, their unbounded outputs can still produce extreme routing controls, resulting in less balanced distributions than Direct. LCAR addresses this issue using the bounded parameterization. It retains the flexibility of a learnable, category-specific λ while constraining its value to a stable range. Consequently, LCAR achieves the lowest CV 0.44, the highest entropy 0.97, and the lowest Top-5 share 39.5%. Compared with Direct, it reduces the CV by approximately 24% and the Top-5 share by 6.9 percentage points. These results show that a learnable λ improves routing adaptability, while bounding it with the sigmoid function further suppresses extreme routing and promotes more balanced utilization of the visual vector bank.

Table 13: Comparison of routing balance under different routing strategies.

	Method	CV ↓	Entropy ↑	Top-5 share ↓
	$\mathcal{S}$-Fixed	0.72	0.93	52.5%
LCAR	Direct	0.58	0.96	46.4%
	Softplus	0.67	0.94	50.5%
	Exponential	0.63	0.95	48.8%
	Sigmoid (ours)	**0.44**	**0.97**	**39.5%**

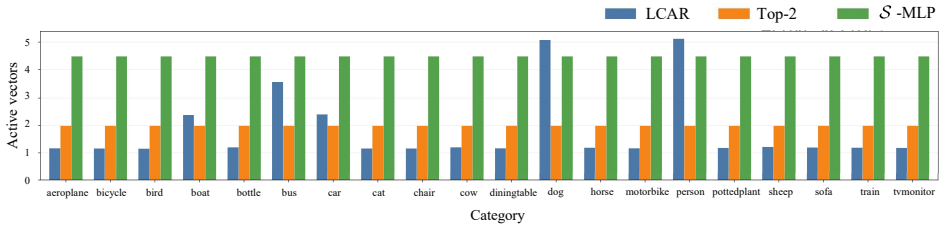


Figure 6: Average Number of Activated Bank Vectors on PV.

Table 14: Effect of HDC Loss Weights.

β_v	0.1	0.3	0.5	0.7	0.9
H	52.7	53.3	53.4	**54.1**	54.0
$\Delta_{\text{ZS}}\downarrow$	1.2	1.4	1.2	0.7	**0.6**
$\bar{F}\downarrow$	6.2	4.4	**3.8**	4.1	3.9

β_c	0.1	0.3	0.5	0.7	0.9
H	53.4	53.8	**54.1**	53.8	54.0
$\Delta_{\text{ZS}}\downarrow$	0.5	**0.3**	0.6	0.4	0.4
$\bar{F}\downarrow$	2.3	1.4	1.1	1.9	**0.8**

β_t	0.1	0.3	0.5	0.7	0.9
H	53.5	53.6	53.4	**54.1**	53.4
$\Delta_{\text{ZS}}\downarrow$	**0.4**	0.7	0.5	**0.4**	0.8
$\bar{F}\downarrow$	1.4	0.6	1.0	**0**	1.0

Category-Adaptive Vector Activation. Fig. 6 compares the average number of activated bank vectors across categories. Both Top-2 and $\mathcal{S}$: L-MLP use category-agnostic activation patterns, resulting in nearly identical numbers of activated vectors across categories. In contrast, LCAR produces substantially different activation patterns across categories, allowing the number of activated bank vectors to vary according to category-specific requirements.

Effect of HDC Loss Weights. We further assess the effect of the auxiliary loss weights, as shown in Table 14. The vector-, category-, and task-level constraints achieve their highest H at $\beta_v = 0.7$, $\beta_w = 0.5$, and $\beta_t = 0.7$, respectively. The results show that increasing the loss weights does not consistently improve knowledge preservation. For the vector-level constraint, $\beta_v = 0.5$ and 0.9 achieve the lowest $\bar{F}$ of 3.8 and Δ_{ZS} of 0.6, respectively. For the category-level constraint, $\beta_w = 0.3$ and 0.9 achieve the lowest Δ_{ZS} of 0.3 and $\bar{F}$ of 0.8, respectively. For the task-level constraint, $\beta_t = 0.1$ and 0.7 achieve the lowest Δ_{ZS} of 0.4, while $\beta_t = 0.7$ achieves the lowest $\bar{F}$ of 0. Based on H, we select $\beta_v = 0.7$, $\beta_w = 0.5$, and $\beta_t = 0.7$.

Class Grouping. To further investigate the effect of category granularity, we explore two strategies for sharing knowledge among related categories: WordNet-based grouping and CLIP-based clustering. For the WordNet-based grouping, it uses the WordNet hierarchy to manually organize all categories into three coarse semantic groups: *animals*, *vehicles*, and *objects*. The resulting groups are as follows:

- **Animals:** fish, jellyfish, penguin, puffin, shark, starfish, stingray, cottontail-rabbit, raccoon, crab, lobster, shrimp, dog, cat, cow, sheep, horse, bird, elephant, bear, zebra, giraffe, chanterelle
- **Vehicles:** boat, car, jetski, aeroplane, airplane, bicycle, bus, motorbike, motorcycle, train, truck, ambulance, skateboard, surfboard, skis, snowboard
- **Objects:** dock, lift, hand, person, package, pistol, pothole, bottle, chair, diningtable, pottedplant, sofa, tvmonitor, traffic light, fire hydrant, stop sign, parking meter, bench, backpack, umbrella, handbag, tie, suitcase, frisbee, sports ball, kite, baseball bat, baseball glove, tennis racket, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, bed, toilet, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

For the CLIP-based clustering, it encodes each class name using CLIP (*ViT-L/14*) Radford et al. (2021) text embeddings and groups semantically similar categories in the embedding space, resulting in six clusters:

- **Cluster 0:** bench, chair, diningtable
- **Cluster 1:** penguin, puffin

Table 16: Per-task performance before and after sequential training. Negative "After−Before" values indicate forgetting, while positive values indicate performance gains.

Stage	Ae	Aq	Co	Eg	Mu	Pa	Pv	Pi	Po	Ra	Sh	Th	Ve	$\bar{F}\downarrow$
Before	35.4	48.5	69.4	72.2	52.9	58.7	70.6	70.3	55.0	74.7	53.0	80.0	69.1	
After	35.0	46.6	73.9	71.2	53.3	63.4	70.5	70.7	50.6	72.1	53.0	80.8	68.5	
After−Before	-0.4	-1.9	+4.5	-1.0	+0.4	+4.7	-0.1	+0.4	-4.4	-2.6	0.0	+0.8	-0.6	0.0167

- **Cluster 2:** aeroplane, ambulance, apple, banana, bear, bed, bicycle, bird, boat, book, bottle, bowl, broccoli, bus, cake, car, carrot, cat, cell phone, chanterelle, clock, cottontail-rabbit, cow, crab, cup, dog, donut, elephant, fish, frisbee, giraffe, hand, horse, hot dog, jellyfish, kite, lift, lobster, motorbike, motorcycle, mouse, orange, oven, package, person, pistol, pizza, pothole, pottedplant, raccoon, refrigerator, sandwich, shark, sheep, shrimp, sports ball, starfish, stingray, stop sign, teddy bear, toilet, traffic light, train, truck, truckperson, tvmonitor, vase, wine glass, zebra
- **Cluster 3:** baseball bat, dock, fire hydrant, fork, jetski, keyboard, knife, laptop, microwave, parking meter, remote, scissors, sink, skateboard, skis, snowboard, spoon, surfboard, tennis racket, tie, toothbrush, umbrella
- **Cluster 4:** backpack, baseball glove, handbag, sofa, suitcase
- **Cluster 5:** hair drier, toaster

The resulting clusters reflect the similarity structure of the CLIP embedding space rather than human-defined semantic boundaries. Compared with WordNet-based grouping, this representation-driven strategy captures implicit visual-semantic relationships and better aligns category organization with the model's feature space.

Table 15: Effect of category organization granularity.

Method	Grouping Strategy	Params (M)	ZCOCO	Avg	H
DitHub	3 groups	17.5	47.0	48.7	47.8
DitHub	6 clusters	27.9	47.4	53.1	50.1
DitHub	Category-level	147.0	47.0	62.2	53.5
DCHA	3 groups	2.7	47.5	53.5	50.3
DCHA	6 clusters	4.1	47.6	56.7	51.8
DCHA	Category-level	20.8	**47.8**	**62.3**	**54.1**

We compare three category organization strategies for DitHub and DCHA in Table 15. Finer granularity consistently improves Avg and H for both methods, supporting the benefit of category-specific adaptation. For DCHA, moving from three groups to six clusters and then to individual categories increases Avg from 53.5 to 56.7 and 62.3, respectively. At the same granularity, DCHA consistently outperforms DitHub while using substantially fewer parameters. Under the three-group and six-cluster settings, DCHA improves Avg by 4.8 and 3.6 points, respectively. At the category level, DCHA achieves 47.8 ZCOCO, 62.3 Avg, and 54.1 H with only 20.8M parameters, compared with 147.0M for DitHub. These results demonstrate that vector-based composition supports effective category specialization with substantially reduced parameter overhead.

Per-Task Performance During Sequential Learning. Table 16 compares performance immediately after each task is learned with performance after the full training sequence. Although we report $\bar{F} = 0$ in the main text after rounding, its unrounded value is 0.0167. This does not mean that no task is forgotten: Po and Ra decline by 4.4 and 2.6 points, respectively. As subsequent tasks are learned, performance on some earlier tasks also improves; for example, Co and Pa gain 4.5 and 4.7 points. These gains offset task-level declines in the average, explaining why the reported forgetting value is nearly zero.

A.6 Metrics for Category Adaptation Analysis

We employ complementary metrics to analyze the inter-category diversity and hierarchical stability of category-specific adaptations. Let $\mathcal{J}$ denote the set of LoRA-injected VLF layers. For each layer $l \in \mathcal{J}$, the effective update of category c is

$$\Delta W_c^l = \gamma B_c^l A_c^l, \tag{18}$$

where A_c^l and B_c^l are constructed by composing vectors from the shared bank through the category-specific routing weights. All reported metrics are averaged across the evaluated VLF layers.

A.6.1 Inter-Category Diversity Metric

We evaluate whether different categories within the same task learn sufficiently diverse adaptation updates using linear centered kernel alignment (CKA). Given two category-specific update matrices $X, Y \in \mathbb{R}^{n\times m}$, we first center them as

$$\widetilde{X} = HX, \qquad \widetilde{Y} = HY, \qquad H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top, \tag{19}$$

where $H \in \mathbb{R}^{n\times n}$ is the centering matrix, I is the identity matrix, and $\mathbf{1} \in \mathbb{R}^n$ is the all-ones vector. Their linear CKA similarity is computed as

$$\mathrm{CKA}(X, Y) = \frac{\left\|\widetilde{X}^\top \widetilde{Y}\right\|_F^2}{\left\|\widetilde{X}^\top \widetilde{X}\right\|_F \left\|\widetilde{Y}^\top \widetilde{Y}\right\|_F}, \tag{20}$$

where $\|\cdot\|_F$ denotes the Frobenius norm and $\top$ denotes matrix transpose. For two categories $c, c' \in \mathcal{C}_t$ from the same task, their category-update similarity is defined as

$$\mathcal{K}_{c,c'} = \frac{1}{|\mathcal{J}|}\sum_{l\in\mathcal{J}} \mathrm{CKA}\left(\Delta W_c^l, \Delta W_{c'}^l\right), \tag{21}$$

where $\mathcal{K}_{c,c'}$ denotes their average structural similarity across the evaluated layers. A larger $\mathcal{K}_{c,c'}$ indicates more similar category-specific updates and thus a higher degree of update homogenization.

A.6.2 Hierarchical Drift Metrics

We quantify accumulated parameter drift at the bank-vector, category-update, and task-update levels using the L1 distance. For a historical task s, the checkpoint immediately after learning task s is fixed as the reference. After learning a subsequent task $t > s$, we compare the current representations with their reference states at task s. For two parameter representations P and Q, their element-wise average L1 distance is defined as

$$d_1(P, Q) = \frac{1}{|\operatorname{vec}(P)|}\left\|\operatorname{vec}(P) - \operatorname{vec}(Q)\right\|_1, \tag{22}$$

where $\operatorname{vec}(\cdot)$ flattens a vector or matrix and $|\operatorname{vec}(P)|$ denotes its number of elements.

Vector distance. Let $\mathcal{I}_s^l$ denote the set of bank vectors activated by the categories of task s at layer l. The bank-vector distance after learning task t is defined as

$$\mathcal{R}_{\mathrm{vec}}^{s\to t} = \frac{1}{|\mathcal{J}|}\sum_{l\in\mathcal{J}} d_1\left(\mathbf{V}^{l,t}\left[\mathcal{I}_s^l\right], \mathbf{V}^{l,s}\left[\mathcal{I}_s^l\right]\right), \tag{23}$$

where $\mathbf{V}^{l,s}$ is the shared vector bank immediately after learning task s, and $\mathbf{V}^{l,t}$ is the bank after learning task t. This metric measures how much the bank vectors previously used by task s change during subsequent training.

Category-update distance. The routing weights of the historical categories are kept fixed when measuring category-update distance. For each category $c \in \mathcal{C}_s$, we use these routing weights to reconstruct $\Delta W_c^{l,s}$ from the reference bank $\mathbf{V}^{l,s}$ and $\Delta W_c^{l,t}$ from the current bank $\mathbf{V}^{l,t}$. The category-update distance is defined as

$$\mathcal{R}_{\mathrm{cat}}^{s\to t} = \frac{1}{|\mathcal{J}||\mathcal{C}_s|}\sum_{l\in\mathcal{J}}\sum_{c\in\mathcal{C}_s} d_1\left(\Delta W_c^{l,t}, \Delta W_c^{l,s}\right), \tag{24}$$

where $\mathcal{C}_s$ denotes the category set of task s. This metric captures the accumulated effect of bank-vector changes on the category-specific updates learned for task s.

Task-update distance. At each evaluation stage, the task-level update of historical task s is obtained by aggregating its reconstructed category-specific updates:

$$\Delta W_s^{l,t} = \frac{1}{|\mathcal{C}_s|} \sum_{c \in \mathcal{C}_s} \Delta W_c^{l,t}, \qquad \Delta W_s^{l,s} = \frac{1}{|\mathcal{C}_s|} \sum_{c \in \mathcal{C}_s} \Delta W_c^{l,s}. \tag{25}$$

The task-update distance is then defined as

$$\mathcal{R}_{\text{task}}^{s \to t} = \frac{1}{|\mathcal{J}|} \sum_{l \in \mathcal{J}} d_1 \left(\Delta W_s^{l,t}, \Delta W_s^{l,s} \right). \tag{26}$$

This metric measures the overall deviation of the adaptation learned for task s after subsequent incremental training. Smaller values of $\mathcal{R}_{\text{vec}}^{s \to t}$, $\mathcal{R}_{\text{cat}}^{s \to t}$, and $\mathcal{R}_{\text{task}}^{s \to t}$ indicate better preservation of the initially learned bank vectors, category updates, and task update, respectively. In our experiments, we set $s = 1$, using the representations obtained after the first task as the fixed reference and measuring their distances after each subsequent task.

A.7 Computation of Routing-Balance Metrics

Let $a_{c,q,i} \in \{0, 1\}$ indicate whether vector i is selected for category c at the q-th LoRA layer. We first calculate the total selection count n of each vector as

$$n_i = \sum_{c \in \mathcal{C}} \sum_{q \in \mathcal{Q}} a_{c,q,i}, \tag{27}$$

where $\mathcal{C}$ denotes the set of evaluated categories and $\mathcal{Q}$ denotes the set of LoRA layers.

Coefficient of variation. CV measures the relative difference among the selection counts of all vectors. Let

$$\bar{n} = \frac{1}{N} \sum_{i=1}^{N} n_i, \qquad s_n = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (n_i - \bar{n})^2}, \tag{28}$$

where N is the shared vector bank size. CV is computed as

$$\text{CV} = \frac{s_n}{\bar{n}}. \tag{29}$$

A lower CV indicates a smaller usage difference among vectors.

Entropy. We normalize the selection counts into a probability distribution:

$$p_i = \frac{n_i}{\sum_{j=1}^{V} n_j}. \tag{30}$$

The normalized entropy is then computed as

$$\text{Entropy} = -\frac{1}{\log V} \sum_{i=1}^{V} p_i \log p_i, \tag{31}$$

where $0 \log 0$ is defined as zero. A higher value indicates that selections are distributed more uniformly across the vector bank.

Top-5 share. Let $n_{(1)} \geq n_{(2)} \geq \cdots \geq n_{(V)}$ denote the selection counts sorted in descending order. The Top-5 share is calculated as

$$\text{Top-5 Share} = \frac{\sum_{i=1}^{5} n_{(i)}}{\sum_{j=1}^{V} n_j}. \tag{32}$$

A lower Top-5 share indicates that the routing decisions are less dominated by the five most frequently selected vectors.