

# Appendix

539

## A Decomposing model inversion objective into layer-wise constraint

541 VMI [41] proposed a model inversion objective based on KL divergence and reformulated it into  
 542 classification and prior terms. In addition to constraining the input prior distribution, DeepInversion  
 543 [51] introduced a constraint on the prior distribution of intermediate layer outputs. Let  $\mathbf{o}_l$  denote the  
 544 output of the  $l$ -th layer,  $L$  the total number of layers,  $\mathbf{x}$  the input, and  $y$  the label, We formulate our  
 545 model inversion objective using KL divergence as:

$$p_s^*(\mathbf{x}) = \arg \min_{p_s(\mathbf{x})} D_{\text{KL}}(p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y) || p_r(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)), \quad (10)$$

546 here  $p_r(\cdot)$  denotes the distribution computed from real data, while  $p_s(\cdot)$  represents the distribution  
 547 from synthetic data. In the following, we show that this objective can be decomposed into a layer-wise  
 548 inversion formulation.

549 Based on the chain rule of probabilities

$$\begin{aligned} p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y) &= p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) p_s(\mathbf{o}_L|y), \\ p_r(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y) &= p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) p_r(\mathbf{o}_L|y), \end{aligned} \quad (11)$$

550 KL divergence in Eq. (10) can be decomposed by

$$\begin{aligned} &D_{\text{KL}}(p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y) || p_r(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)) \\ &= \mathbb{E}_{p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)} \log \frac{p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)}{p_r(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)} \\ &= \mathbb{E}_{p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) p_s(\mathbf{o}_L|y)} \log \frac{p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) p_s(\mathbf{o}_L|y)}{p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) p_r(\mathbf{o}_L|y)} \\ &= \int \dots \int p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) p_s(\mathbf{o}_L|y) \left( \log \frac{p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)}{p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)} + \log \frac{p_s(\mathbf{o}_L|y)}{p_r(\mathbf{o}_L|y)} \right) d\mathbf{x} \dots d\mathbf{o}_L \\ &= \int p_s(\mathbf{o}_L|y) \left( \int \dots \int p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) \log \frac{p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)}{p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)} d\mathbf{x} \dots d\mathbf{o}_{L-1} \right) d\mathbf{o}_L \\ &\quad + \int \dots \int p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) \left( \int p_s(\mathbf{o}_L|y) \log \frac{p_s(\mathbf{o}_L|y)}{p_r(\mathbf{o}_L|y)} d\mathbf{o}_L \right) d\mathbf{x} \dots d\mathbf{o}_{L-1} \\ &= \mathbb{E}_{p_s(\mathbf{o}_L|y)} D_{\text{KL}}(p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) || p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)) \\ &\quad + \mathbb{E}_{p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)} D_{\text{KL}}(p_s(\mathbf{o}_L|y) || p_r(\mathbf{o}_L|y)). \end{aligned} \quad (12)$$

551 The KL divergence in the second term of Eq. (12) is not included in the inte-  
 552 gration over  $\mathbf{o}_{L-1}, \dots, \mathbf{x}$ , and the expectation simplifies to the KL divergence itself:  
 553  $\mathbb{E}_{p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)} D_{\text{KL}}(p_s(\mathbf{o}_L|y) || p_r(\mathbf{o}_L|y)) = D_{\text{KL}}(p_s(\mathbf{o}_L|y) || p_r(\mathbf{o}_L|y))$ . We further assume that  
 554 inversion through each layer retains essential information. Specifically,  $\mathbf{o}_L$  is assumed to contain all  
 555 information about the label  $y$ , and the representations  $\mathbf{x}, \dots, \mathbf{o}_{L-1}$  can be recovered from  $\mathbf{o}_L$ . This  
 556 assumption implies a conditional independence relationship, which can be formulated as:

$$p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) \approx p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L), \quad p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) \approx p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L). \quad (13)$$

557 Therefore, the decomposed objective is:

$$\begin{aligned} &D_{\text{KL}}(p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y) || p_r(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)) \\ &= \mathbb{E}_{p_s(\mathbf{o}_L|y)} D_{\text{KL}}(p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y) || p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)) \\ &\quad + \mathbb{E}_{p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L, y)} D_{\text{KL}}(p_s(\mathbf{o}_L|y) || p_r(\mathbf{o}_L|y)) \\ &\approx \mathbb{E}_{p_s(\mathbf{o}_L|y)} D_{\text{KL}}(p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L) || p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L)) + D_{\text{KL}}(p_s(\mathbf{o}_L|y) || p_r(\mathbf{o}_L|y)). \end{aligned} \quad (14)$$

558 The expectation on KL divergence of the right hand side can be decomposed with the same method  
 559 as:

$$\begin{aligned}
& \mathbb{E}_{p_s(\mathbf{o}_L|y)} [D_{\text{KL}}(p_s(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L)||p_r(\mathbf{o}_{L-1}, \dots, \mathbf{x}|\mathbf{o}_L))] \\
& \approx \mathbb{E}_{p_s(\mathbf{o}_L|y)} [\mathbb{E}_{p_s(\mathbf{o}_{L-1}|\mathbf{o}_L)} D_{\text{KL}}(p_s(\mathbf{o}_{L-2}, \dots, \mathbf{x}|\mathbf{o}_{L-1})||p_r(\mathbf{o}_{L-2}, \dots, \mathbf{x}|\mathbf{o}_{L-1}))] \\
& \quad + \mathbb{E}_{p_s(\mathbf{o}_L|y)} D_{\text{KL}}(p_s(\mathbf{o}_{L-1}|\mathbf{o}_L)||p_r(\mathbf{o}_{L-1}|\mathbf{o}_L)) \\
& \approx \mathbb{E}_{p_s(\mathbf{o}_{L-1}|y)} [D_{\text{KL}}(p_s(\mathbf{o}_{L-2}, \dots, \mathbf{x}|\mathbf{o}_{L-1})||p_r(\mathbf{o}_{L-2}, \dots, \mathbf{x}|\mathbf{o}_{L-1}))] \\
& \quad + \mathbb{E}_{p_s(\mathbf{o}_L|y)} D_{\text{KL}}(p_s(\mathbf{o}_{L-1}|\mathbf{o}_L)||p_r(\mathbf{o}_{L-1}|\mathbf{o}_L)).
\end{aligned} \tag{15}$$

560 The last line in Eq. (15) is based on conditional independence, namely

$$\begin{aligned}
p_s(\mathbf{o}_{L-1}|\mathbf{o}_L, y) & \approx p_s(\mathbf{o}_{L-1}|\mathbf{o}_L) \\
p_s(\mathbf{o}_{L-1}|y) & = \int p_s(\mathbf{o}_{L-1}|\mathbf{o}_L, y)p_s(\mathbf{o}_L|y)d\mathbf{o}_L \\
& \approx \int p_s(\mathbf{o}_{L-1}|\mathbf{o}_L)p_s(\mathbf{o}_L|y)d\mathbf{o}_L.
\end{aligned} \tag{16}$$

561 The first term in the right hand side of Eq. (15) can be decomposed in the same way as:

$$\begin{aligned}
& \mathbb{E}_{p_s(\mathbf{o}_{L-1}|y)} [D_{\text{KL}}(p_s(\mathbf{o}_{L-2}, \dots, \mathbf{x}|\mathbf{o}_{L-1})||p_r(\mathbf{o}_{L-2}, \dots, \mathbf{x}|\mathbf{o}_{L-1}))] \\
& \approx \mathbb{E}_{p_s(\mathbf{o}_{L-1}|y)} [\mathbb{E}_{p_s(\mathbf{o}_{L-2}|\mathbf{o}_{L-1})} D_{\text{KL}}(p_s(\mathbf{o}_{L-3}, \dots, \mathbf{x}|\mathbf{o}_{L-2})||p_r(\mathbf{o}_{L-3}, \dots, \mathbf{x}|\mathbf{o}_{L-2}))] \\
& \quad + \mathbb{E}_{p_s(\mathbf{o}_{L-1}|y)} D_{\text{KL}}(p_s(\mathbf{o}_{L-2}|\mathbf{o}_{L-1})||p_r(\mathbf{o}_{L-2}|\mathbf{o}_{L-1})) \\
& \approx \mathbb{E}_{p_s(\mathbf{o}_{L-2}|y)} [D_{\text{KL}}(p_s(\mathbf{o}_{L-3}, \dots, \mathbf{x}|\mathbf{o}_{L-2})||p_r(\mathbf{o}_{L-3}, \dots, \mathbf{x}|\mathbf{o}_{L-2}))] \\
& \quad + \mathbb{E}_{p_s(\mathbf{o}_{L-1}|y)} D_{\text{KL}}(p_s(\mathbf{o}_{L-2}|\mathbf{o}_{L-1})||p_r(\mathbf{o}_{L-2}|\mathbf{o}_{L-1})).
\end{aligned} \tag{17}$$

562 Given  $\mathbf{o}_0 = \mathbf{x}$  and  $\mathbf{o}_{L+1} = y$ , decomposing inversion objective layer-by-layer recursively results in

$$\begin{aligned}
& D_{\text{KL}}(p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)||p_r(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, \mathbf{x}|y)) \\
& \approx \sum_{l=1}^L \mathbb{E}_{p_s(\mathbf{o}_{l+1}|y)} [\mathbb{E}_{p_s(\mathbf{o}_l|\mathbf{o}_{l+1})} D_{\text{KL}}(p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)||p_r(\mathbf{o}_{l-1}|\mathbf{o}_l))] + D_{\text{KL}}(p_s(\mathbf{o}_L|y)||p_r(\mathbf{o}_L|y)).
\end{aligned} \tag{18}$$

563 Eq. (18) shows that, under the conditional independence assumption, the input to each layer can be  
564 inferred from its output. Summing the layer-wise KL divergences thus provides an approximation of  
565 the total inversion objective in Eq. (10).

## 566 B Layer-wise and full-model inversion objective

567 The KL divergences in Eq. (18) are intractable in practice. Following [41], we reformulate them  
568 using Bayes' rule as:

$$\begin{aligned}
& D_{\text{KL}}(p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)||p_r(\mathbf{o}_{l-1}|\mathbf{o}_l)) \\
& = \mathbb{E}_{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)} \log \frac{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)}{p_r(\mathbf{o}_{l-1}|\mathbf{o}_l)} \\
& = \mathbb{E}_{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)} \log \frac{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)p_r(\mathbf{o}_l)}{p_r(\mathbf{o}_l|\mathbf{o}_{l-1})p_r(\mathbf{o}_{l-1})} \\
& = \mathbb{E}_{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)} \log \frac{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)}{p_r(\mathbf{o}_{l-1})} - \mathbb{E}_{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)} \log p_r(\mathbf{o}_l|\mathbf{o}_{l-1}) + \log p_r(\mathbf{o}_l) \\
& = D_{\text{KL}}(p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)||p_r(\mathbf{o}_{l-1})) - \mathbb{E}_{p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)} \log p_r(\mathbf{o}_l|\mathbf{o}_{l-1}) + \log p_r(\mathbf{o}_l).
\end{aligned} \tag{19}$$

569 Since  $p_r(\mathbf{o}_l)$  is computed from real data and is independent of the synthetic data, we treat it as a  
570 constant. The negative log-probability term  $-\log p_r(\mathbf{o}_l|\mathbf{o}_{l-1})$  is approximated using mean squared  
571 error (MSE). For the KL divergence at the final layer  $D_{\text{KL}}(p_s(\mathbf{o}_L|y)||p_r(\mathbf{o}_L|y))$ , the negative log-  
572 probability  $-\log p_r(y|\mathbf{o}_L)$  can be approximated using cross-entropy (CE) loss for classification tasks,  
573 and mean squared error (MSE) for regression tasks. In this work, we adopt the CE loss to construct  
574 our final inversion objective. For the prior constraint  $D_{\text{KL}}(p_s(\mathbf{o}_{l-1}|\mathbf{o}_l)||p_r(\mathbf{o}_{l-1}))$ , we follow ABD  
575 [34] by approximating both distributions as Gaussians and computing the KL divergence in closed  
576 form.

577 The expectation  $\mathbb{E}_{p_s(\mathbf{o}_{l+1}|y)} [\mathbb{E}_{p_s(\mathbf{o}_l|\mathbf{o}_{l+1})} D_{\text{KL}}(p_s(\mathbf{o}_{l-1}|\mathbf{o}_l) || p_r(\mathbf{o}_{l-1}|\mathbf{o}_l))]$  can be approximated by  
 578 averaging the KL divergences computed over a batch of  $\mathbf{o}_l$  if  $\mathbf{o}_l$  is known. Building on the layer-wise  
 579 constraint in Eq. (18) and the approximations above,  $\mathbf{o}_L$  can be optimized with given  $y$  using objective

$$D_{\text{KL}}(p_s(\mathbf{o}_L|y) || p_r(\mathbf{o}_L|y)) \approx D_{\text{KL}}(\mathcal{N}(\hat{\mu}_L, \hat{\sigma}_L) || \mathcal{N}(\mu_L, \sigma_L)) + \frac{1}{N} \sum_{i=1}^N \ell_{\text{CE}}(\mathbf{o}_{L,i}, y; \boldsymbol{\theta}_{L+1}). \quad (20)$$

580 Therefore, we propose a top-down optimization strategy that optimize features layer-by-layer from  
 581 the output layer to the input. At each step, the optimized  $\mathbf{o}_{l+1}$  is used as the target output to guide  
 582 the optimization of  $\mathbf{o}_l$ . In other words,  $\mathbf{o}_l$  is optimized before proceeding to  $\mathbf{o}_{l-1}$ . The layer-wise  
 583 inversion objective is

$$\mathbb{E}_{p_s(\mathbf{o}_{l+1}|y)} [\mathbb{E}_{p_s(\mathbf{o}_l|\mathbf{o}_{l+1})} D_{\text{KL}}(p_s(\mathbf{o}_{l-1}|\mathbf{o}_l) || p_r(\mathbf{o}_{l-1}|\mathbf{o}_l))]$$

$$\approx D_{\text{KL}}(\mathcal{N}(\hat{\mu}_{l-1}, \hat{\sigma}_{l-1}) || \mathcal{N}(\mu_{l-1}, \sigma_{l-1})) + \frac{1}{N} \sum_{i=1}^N \ell_{\text{MSE}}(\mathbf{o}_{l-1,i}, \mathbf{o}_{l,i}; \boldsymbol{\theta}_l), \quad (21)$$

584 where  $\boldsymbol{\theta}_l$  denotes the parameters of the  $l$ -th layer,  $\mu_{l-1}$  and  $\sigma_{l-1}$  are the mean and standard deviation  
 585 computed from real data, and  $\hat{\mu}_{l-1}$  and  $\hat{\sigma}_{l-1}$  are the corresponding statistics computed from synthetic  
 586 data. The model inversion objective over all layers is given as:

$$D_{\text{KL}}(p_s(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, x|y) || p_r(\mathbf{o}_L, \mathbf{o}_{L-1}, \dots, x|y))$$

$$\approx \sum_{l=1}^L \left( D_{\text{KL}}(\mathcal{N}(\hat{\mu}_{l-1}, \hat{\sigma}_{l-1}) || \mathcal{N}(\mu_{l-1}, \sigma_{l-1})) + \frac{1}{N} \sum_{i=1}^N \ell_{\text{MSE}}(\mathbf{o}_{l-1,i}, \mathbf{o}_{l,i}; \boldsymbol{\theta}_l) \right)$$

$$+ D_{\text{KL}}(\mathcal{N}(\hat{\mu}_L, \hat{\sigma}_L) || \mathcal{N}(\mu_L, \sigma_L)) + \frac{1}{N} \sum_{i=1}^N \ell_{\text{CE}}(\mathbf{o}_{L,i}, y; \boldsymbol{\theta}_{L+1}). \quad (22)$$

587 Performing model inversion on the full model omits the loss terms  $\ell_{\text{MSE}}(\mathbf{o}_{l-1,i}, \mathbf{o}_{l,i}; \boldsymbol{\theta}_l)$ , as there  
 588 is no available target  $\mathbf{o}_{l,i}$  for the output of the  $l$ -th layer. Equation (22) is equivalent to the model  
 589 inversion objective proposed in ABD [34], excluding the smoothness constraint. Since the loss  
 590 landscape of a single layer is much simpler than that of the full model, inversion at the layer level  
 591 requires significantly fewer update steps.

## 592 C Detailed algorithms

### 593 C.1 Implementation details on ResNets-based CL

594 Since ResNets are trained from scratch in continual learning, the feature representations of previous  
 595 classes may shift after learning new tasks. To account for this, we update the mean and standard  
 596 deviation of previous classes using synthetic data after each task. The contrastive models for each  
 597 previous class are then retrained accordingly. Once the class-wise statistics and contrastive models  
 598 are updated, we sample features for model inversion.

599 For continual learning with ResNets, we enhance the separability of features from new tasks by  
 600 incorporating classification layer fine-tuning, following ABD [34], and based on the replay loss from  
 601 R-DFCIL [10]. The loss function used during training on task  $t$  is:

$$\mathcal{L}_{\text{CL}}(\boldsymbol{\theta}_t) = \frac{1}{N} \sum_{i=1}^N \ell_{\text{ICE}}(\mathbf{x}_i, y_i; \boldsymbol{\theta}_t) + \frac{1}{M} \sum_{j=1}^M (\lambda_{\text{hkd}} \mathcal{L}_{\text{hkd}}(\mathbf{x}_j; \boldsymbol{\theta}_t, \boldsymbol{\theta}_{t-1}) + \lambda_{\text{rkd}} \mathcal{L}_{\text{rkd}}(\mathbf{x}_j; ))$$

$$+ \lambda_{\text{ft}} \frac{1}{M+N} \sum_{k=1}^{M+N} \mathcal{L}_{\text{ft}}(\mathbf{x}_k, y_k; \boldsymbol{\theta}_t), \quad (23)$$

602 where  $\ell_{\text{ICE}}$  denotes the local cross-entropy loss, computed only over the classes of the current task.  
 603  $\mathcal{L}_{\text{ft}}$ , proposed in [34], is a cross-entropy loss over all previously seen classes, and is used exclusively  
 604 to update the classification layer.  $\mathcal{L}_{\text{hkd}}$  and  $\mathcal{L}_{\text{rkd}}$  refer to the hard knowledge distillation (HKD) loss

and the relational knowledge distillation (RKD) loss, respectively, both introduced in [10].  $\theta_t$  denotes the parameters of the CL model at task  $t$ , and  $\theta_{t-1}$  represents the parameters after training on the previous task.  $\lambda_{\text{hkd}}$ ,  $\lambda_{\text{rkd}}$ , and  $\lambda_{\text{ft}}$  are hyperparameters that weight their corresponding loss terms to balance their contributions during optimization, and we apply the loss factor change scheme proposed in R-DFCIL. Detailed algorithm is presented in Alg. 3, and analysis on the effect of  $\mathcal{L}_{\text{ft}}$  is provided in Appendix D.1.

---

**Algorithm 3: ResNet-Based CL**

---

**Input:** Dataset sequence  $\mathcal{D}_{1:T}$

**Output:** Trained CL model parameters  $\theta_T$

Initialize model parameter  $\theta_0$ ;

**for**  $t = 1$  **to**  $T$  **do**

**if**  $i > 1$  **then**

        Sample features for inversion;

        PMI+full-model inversion to generate previous data;

        Train model with replay;

**else**

        Train model with CE loss;

    Finetune classification layer;

    Compute class-wise feature Gaussian distribution for new classes;

    Update class-wise feature Gaussian distribution for old classes;

    Train contrastive model for each seen class;

    Save model as teacher model;

---

610

611 **C.2 Implementation details on CLIP-based CL**

Since CLIP models encode rich pre-trained knowledge, their image features already contain strong semantic information for classification. We therefore assume that the features of previously learned classes do not drift significantly. As a result, we do not update the feature statistics or contrastive models for previous classes in CLIP-based CL. For all our methods, classification is performed using local cross-entropy loss, computed only on the current task classes.

We integrate our data generation method into three baseline approaches: VPT [16], CODA-Prompt [35], and MoE-Adapter [52]. VPT introduces learnable prompts into the vision encoder while keeping the text encoder frozen. CODA-Prompt extends the learnable prompts in the image encoder after each task and uses a learnable classification head. For both methods, we apply only the hard knowledge distillation (HKD) loss on synthetic data during replay to mitigate forgetting. MoE-Adapter incorporates a Mixture of Experts (MoE) into both the image and text encoders. To prevent forgetting on text encoder, we introduce a text knowledge distillation loss to prevent forgetting in the text encoder by

$$\mathcal{L}_{\text{tkd}}(\mathbf{t}_{ci}; F_{t,k-1}, F_{t,k}) = \|F_{t,k-1}(\mathbf{t}_{ci}) - F_{t,k}(\mathbf{t}_{ci})\|_1, \quad (24)$$

where  $F_{t,k}$  denotes the text encoder during training on the  $k$ -th task, and  $F_{t,k-1}$  refers to the text encoder after training on the previous task. Additionally, we introduce a text encoder fine-tuning loss to improve classification performance over all previously seen classes, defined as:

$$\mathcal{L}_{\text{ft}}(\mathbf{x}_{ci}, \mathbf{t}_{ci}; F_i, F_t) = \mathcal{L}_{CE}(F_i(\mathbf{x}_{ci}), F_t(\mathbf{t}_{ci})). \quad (25)$$

The text encoder fine-tuning loss is used exclusively to update the text encoder for improved classification performance over all seen classes. Loss factor changing scheme proposed in R-DFCIL is also applied in our CLIP-based CL method. Detailed algorithm is shown in Alg. 4.

In practice, we found that performing model inversion through the first convolutional layer of the ViT model significantly degrades continual learning performance. This is because the layer uses a kernel size of 16 and a stride of 16, leading to substantial information loss from the input. Moreover, this layer remains frozen during continual learning. To address this, we implement model inversion to generate the output features of the first convolutional layer and apply this approach across all CLIP-based continual learning experiments.

### C.3 Model inversion details

Based on the model inversion loss in Eq. (27), prior works [18, 51, 34] introduce a smoothness constraint on the synthetic data. In our proposed PMI method, we omit this constraint, as enforcing smoothness may result in the loss of important information in the feature maps.

For full-model inversion, we find that removing the smoothness constraint does not affect continual learning performance in ResNet-based experiments. Therefore, we omit it in this setting. In CLIP-based continual learning experiments, we perform model inversion to generate the output feature map of the first convolutional layer. To emulate the smoothness constraint typically applied to input images, we apply a total variation loss to this feature map, with a fixed weight of  $5 \times 10^{-3}$  for all CL experiments. For visualization experiments, we follow the same setup and apply total variation loss with the same weight, consistent with the setting in [18].

The prior distribution constraint in Eq. (21) requires layer-wise statistics of real data. Unlike ResNets, the ViT backbone used in CLIP does not include batch normalization layers. To apply the prior distribution constraint to the CLIP model, we compute the mean and standard deviation over real data after training each task and maintain a moving average across all previously seen tasks. In practice, we treat each residual block as a single layer in ResNet architectures, and each residual transformer block as a single layer in the ViT backbone of the CLIP model.

In practice, we introduce a scaling factor for each loss term in both the PMI objective and the full-model inversion objective to balance their relative contributions. Specifically, the layer-wise inversion objective is given by:

$$\mathcal{L}_{\text{inv}}(\mathbf{o}_{l-1}; \boldsymbol{\theta}_l) = \alpha_l D_{\text{KL}}(\mathcal{N}(\hat{\mu}_{l-1}, \hat{\sigma}_{l-1}) || \mathcal{N}(\mu_{l-1}, \sigma_{l-1})) + \beta_l \frac{1}{N} \sum_{i=1}^N \ell_{\text{MSE}}(\mathbf{o}_{l-1,i}, \mathbf{o}_{l,i}; \boldsymbol{\theta}_l), \quad (26)$$

where  $\alpha_l$  and  $\beta_l$  are the scaling factors for the prior distribution constraint and the output constraint at layer  $l$ , respectively. The overall model inversion loss across all layers is then defined as:

$$\begin{aligned} \mathcal{L}_{\text{inv}}(\mathbf{x}; \boldsymbol{\theta}) = & \sum_{l=1}^L \left( \alpha_l D_{\text{KL}}(\mathcal{N}(\hat{\mu}_{l-1}, \hat{\sigma}_{l-1}) || \mathcal{N}(\mu_{l-1}, \sigma_{l-1})) + \beta_l \frac{1}{N} \sum_{i=1}^N \ell_{\text{MSE}}(\mathbf{o}_{l-1,i}, \mathbf{o}_{l,i}; \boldsymbol{\theta}_l) \right) \\ & + \alpha_{L+1} D_{\text{KL}}(\mathcal{N}(\hat{\mu}_L, \hat{\sigma}_L) || \mathcal{N}(\mu_L, \sigma_L)) + \beta_{L+1} \frac{1}{N} \sum_{i=1}^N \ell_{\text{CE}}(\mathbf{o}_{L,i}, y; \boldsymbol{\theta}_{L+1}) + \gamma \mathcal{L}_{\text{tv}}(\mathbf{x}), \end{aligned} \quad (27)$$

where  $\mathcal{L}_{\text{tv}}$  denotes the total variation loss used to enforce smoothness,  $\gamma$  is its corresponding weighting factor, and  $\boldsymbol{\theta}$  represents the model parameters.

## D Additional experiment results

### D.1 ResNet-based CL

To further demonstrate the effectiveness of our method, we include DCMi [29] in our comparison and report the final average accuracy in Table 5. To specifically evaluate the impact of our PMI + full-model inversion strategy and feature modeling approach, we additionally incorporate the classification head fine-tuning loss into R-DFCIL and compare the results with our method in terms of final average accuracy on CIFAR-100 dataset, as shown in Table 6. The variant *R-DFCIL+* denotes

Table 5: Final average accuracies on CIFAR-100 and Tiny-ImageNet using a ResNet-32 backbone, including the DCMI baseline. Red and blue values indicate the best and second-best performance, respectively. Our method consistently outperforms all existing baselines across all settings.

Method	CIFAR-100			Tiny-ImageNet		
	5 task	10 task	20 task	5 task	10 task	20 task
Upper bound	70.59±0.14	70.59±0.14	70.59±0.14	55.25±0.41	55.25±0.41	55.25±0.41
DeepInversion	20.48±1.11	11.26±0.46	5.63±0.10	-	-	-
ABD	48.84±0.33	36.75±0.45	24.40±0.60	30.83±0.46	23.17±0.45	14.61±0.47
DCMI	41.05±0.67	27.70±1.07	18.09±0.85	35.78±0.37	25.89±0.17	17.03±0.08
R-DFCIL	49.87±0.45	41.80±0.24	31.54±0.54	35.33±0.02	29.05±0.28	24.85±0.16
<b>Ours w/o CFS</b>	<b>52.05±0.02</b>	<b>43.23±0.28</b>	<b>32.23±0.42</b>	<b>37.65±0.24</b>	<b>32.09±0.20</b>	<b>25.51±0.56</b>
<b>Ours</b>	<b>52.38±0.53</b>	<b>43.90±0.35</b>	<b>32.60±0.29</b>	<b>37.90±0.10</b>	<b>32.43±0.09</b>	<b>25.67±0.71</b>

Table 7: CLIP performance on CIFAR-100 and ImageNet-R using a CLIP model pre-trained on LAION-400M. Numbers in parentheses indicate the absolute improvement over the corresponding baseline methods. Our method consistently enhances the performance of all baselines on both datasets.

Method	No Real Image Buffer	CIFAR-100		ImageNet-R	
		Avg.	Last	Avg.	Last
iCaRL	✗	79.91	63.94	72.22	54.38
MEMO	✗	84.67	74.98	80.00	74.07
PROOF	✗	86.70	79.05	85.34	80.10
ZS-CLIP	✓	81.38	71.26	82.93	76.67
VPT	✓	84.81	74.75	84.99	79.45
CODA-P	✓	85.00	76.56	83.70	75.73
MoE-Adapter	✓	88.01	79.97	85.75	80.83
<b>Ours + VPT</b>	✓	<b>86.24</b> (1.43)	<b>76.13</b> (1.38)	<b>85.87</b> (0.88)	<b>80.18</b> (0.73)
<b>Ours + CODA-P</b>	✓	<b>85.84</b> (0.84)	<b>77.66</b> (1.10)	<b>84.82</b> (1.12)	<b>76.15</b> (0.42)
<b>Ours + MoE-Adapter</b>	✓	<b>88.55</b> (0.54)	<b>80.51</b> (0.54)	<b>87.18</b> (1.43)	<b>82.48</b> (1.65)

679 R-DFCIL with the classification layer fine-tuning loss, i.e., the loss function is identical to that used  
680 in our method. All experiments follow the same setup as described in Section 4.1.

681 As shown in Table 5, our method continues to outperform other baselines even after including DCMI  
682 in the comparison, further demonstrating its effectiveness. In Table 6, incorporating the classification  
683 head fine-tuning loss slightly improves the performance of R-DFCIL; however, our method still  
684 surpasses the *R-DFCIL+* variant. This suggests that the performance gain primarily stems from  
685 our PMI+full-model inversion strategy and the proposed CFS method. These results provide strong  
686 evidence of the effectiveness of our approach.

## 687 D.2 CLIP-based CL with LAION-400M pre-trained weight

688 To further demonstrate the robust-  
689 ness of our method, we conduct ad-  
690 ditional experiments using CLIP mod-  
691 els pretrained on LAION-400M, eval-  
692 uated on CIFAR-100 and ImageNet-  
693 R. Our method is integrated with VPT  
694 [16], CODA-Prompt [35], and MoE-  
695 Adapter [52]. All hyperparameters are  
696 kept consistent with those used in Sec-  
697 tion 4.1. The final average accuracy  
698 and the average accuracy across all incremental stages are reported in Table 7. Our method consis-  
699 tently improves the performance of all baseline methods on both datasets, further demonstrating its  
700 effectiveness, robustness, and compatibility.

Table 6: Ablation study on classification head finetuning loss on CIFAR-100 dataset.

Method	5 task	10 task	20 task
R-DFCIL	49.87±0.45	41.80±0.24	31.54±0.54
R-DFCIL+	50.45±0.21	41.93±0.56	31.19±0.26
<b>Ours w/o CFS</b>	<b>52.05±0.02</b>	<b>43.23±0.28</b>	<b>32.23±0.42</b>
<b>Ours</b>	<b>52.38±0.53</b>	<b>43.90±0.35</b>	<b>32.60±0.29</b>

## E Feature visualization

**Distribution shift between real data and synthetic data.** Previous works [51, 34, 10] use cross-entropy loss in the model inversion objective. In the experiment on the CIFAR-100 dataset with 10 tasks, experiment settings are kept the same as experiments in Section 4.1. We visualize the features of real and synthetic data from the new classes after training each task using t-SNE, as shown in Figure 5. The synthetic data is generated by method in R-DFCIL. In the plot, synthetic features are represented by circular dots, while real data features are shown as translucent triangular dots.

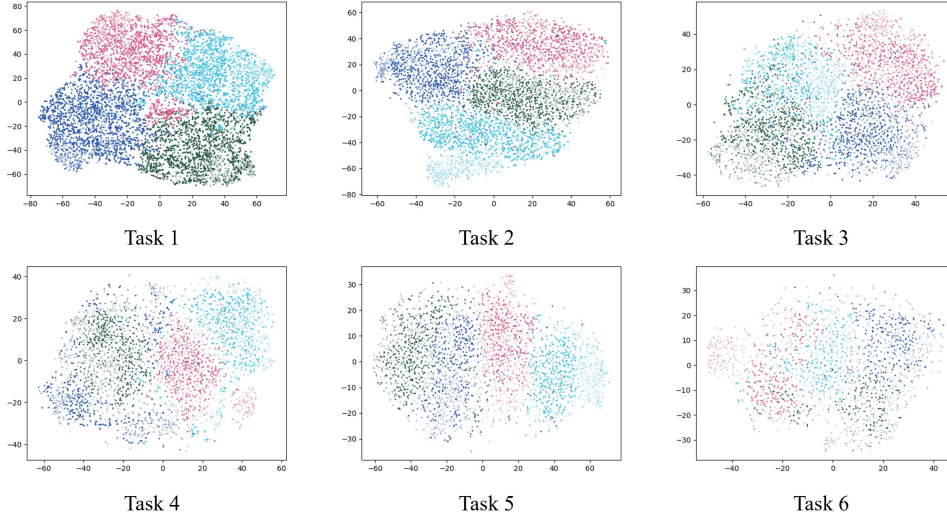


Figure 5: t-SNE visualization of features computed from synthetic data (circular dots) and real data (translucent triangular dots) from four classes in each of the first six tasks. A noticeable distribution shift can be observed between the real and synthetic features.

As shown in Figure 5, an apparent distribution shift between real and synthetic features is observed even on newly trained tasks. This shift suggests that the synthetic data may encode different information than the real data.

**Inversion from feature modeling.** To demonstrate the effectiveness of our approach for sampling features from class-wise distributions for model inversion, we additionally visualize feature t-SNE under the same experimental setting in Section 4.1, using features sampled from class-wise Gaussian distributions, as shown in Figure 6. The distributions of real and synthetic features are more consistent across tasks, indicating that our method significantly mitigates the distribution shift problem.

To further demonstrate the effectiveness of CFS based on Gaussian distributions, we generate synthetic samples using features sampled from Gaussian and Gaussian+CFS distributions, respectively, and visualize the features of four classes from the last three tasks in Figure 7. The first row of Figure 7 corresponds to sampling from class-wise Gaussian distributions for model inversion, while the second row shows results with class-wise Gaussian distributions combined with CFS. Real features are represented by translucent triangular dots.

The synthetic features in the second row more closely align with the real class-wise features compared to those in the first row, illustrating the effectiveness of our CFS method. We acknowledge that t-SNE visualizations may not fully reflect the impact of CFS due to the significant dimensionality reduction involved. However, the results presented in Section 4.1 and Section 4.2 clearly demonstrate the effectiveness of CFS in improving continual learning performance.

## F Comparison with other model inversion methods

To evaluate the effectiveness of our method, we implement Sparse Model Inversion (SMI) [13] in a CLIP-based continual learning setting and integrate it with MoE-Adapter [52]. We compare our method against SMI and full-model inversion in terms of continual learning performance and computational cost. All experimental settings are kept consistent with those in Section 4.1, with CFS

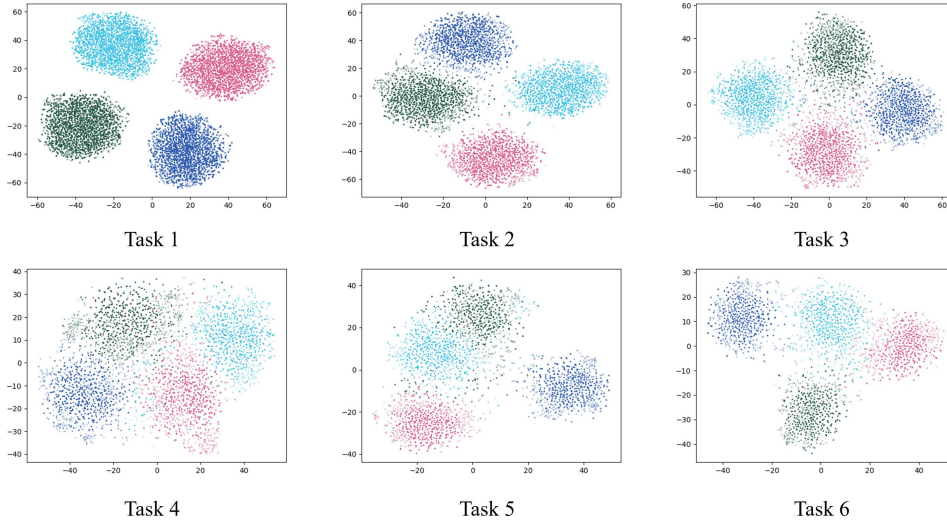


Figure 6: t-SNE visualization of features computed from synthetic data (circular dots) and real data (translucent triangular dots) from four classes in each of the first six tasks. The synthetic data is generated using features sampled from class-wise Gaussian distributions. The resulting feature distributions of synthetic and real data are more consistent.

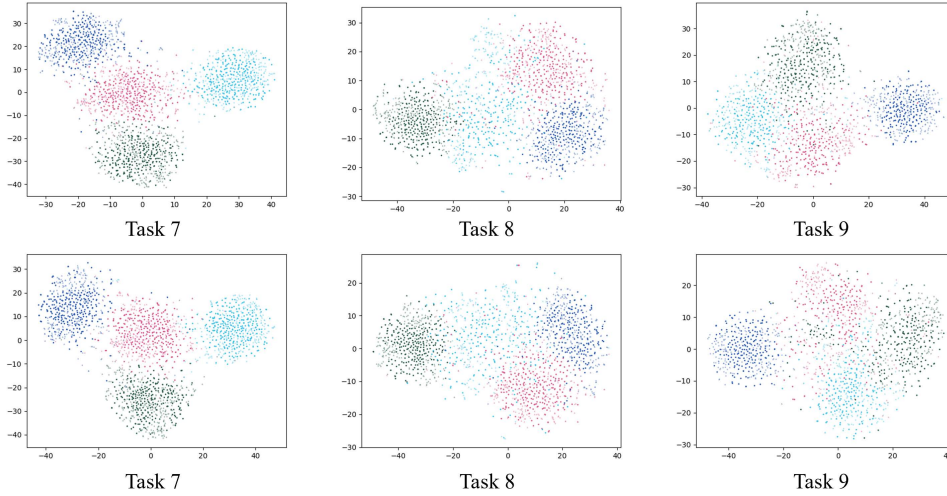


Figure 7: t-SNE visualizations of synthetic features from classes in the last three tasks. The first row shows features sampled from class-wise Gaussian distributions for model inversion, while the second row incorporates class-wise Gaussian sampling with CFS. In both cases, the synthetic and real feature distributions are more consistent, with the synthetic features in the second row more closely covering the real feature distribution.

732 applied across all experiments. The results are presented in Table 8. For full-model inversion, we  
733 follow the settings of [18] and set the number of update steps to 3,400. For SMI, we adopt the original  
734 configuration with 4,000 update steps and a pruning ratio of 76%. All experiments are conducted on  
735 NVIDIA GeForce RTX 3090 GPUs with 24 GB of memory, using an Intel(R) Core(TM) i9-12900K  
736 CPU.

737 Our PMI combined with full-model inversion consistently outperforms other model inversion methods  
738 in continual learning performance while incurring the lowest time cost, demonstrating the effective-  
739 ness of our approach. While SMI achieves comparable time efficiency, excessive patch pruning can  
740 lead to a performance drop compared to full-model inversion. The time cost for generating a single  
741 sample further highlights the efficiency of our method.



Table 8: CL performance and computational cost of different model inversion methods integrated with MoE-Adapter on the CIFAR-100 dataset. Our method achieves the best performance while incurring the lowest time cost.

Method	Avg	Last	Time cost	Per-image time cost
Full-model+MoE-Adapter	88.13	80.53	17h 17m	23.15s
SMI+MoE-Adapter	88.16	80.46	7h 14m	13.35s
Ours+MoE-Adapter	<b>88.35</b>	<b>81.06</b>	<b>6h 10m</b>	<b>7.42s</b>

## 742 G Loss landscape visualization

743 To support our claim that the inversion loss landscape of a single layer is simpler than that of the  
744 full model, we implement our method on the image encoder (ViT-B/16) of the CLIP model and  
745 visualize the inversion loss landscape of each layer and the full model near the optimal point, as  
746 shown in Figure 8. The inversion loss landscapes of individual layers are significantly simpler and  
747 flatter compared to that of the full model. As a result, performing inversion through a single layer  
748 enables faster convergence and achieves lower inversion loss.

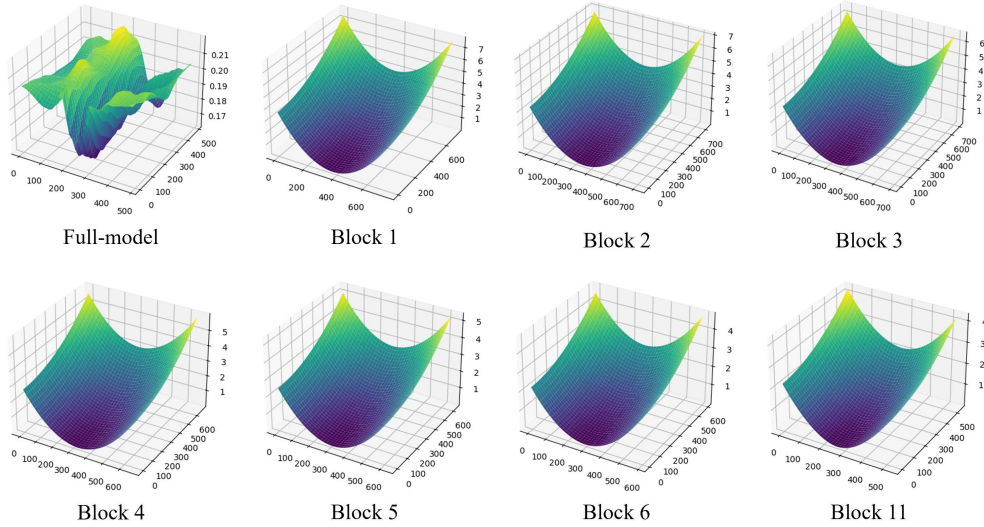


Figure 8: Inversion loss landscapes of full-model fine-tuning and individual layers. The loss landscape of a single layer is significantly simpler than that of the full model.

## 749 H Generating image from text using model inversion on CLIP model

750 To visually demonstrate the effectiveness of our PMI+full-model inversion strategy, we follow the  
751 experimental setup of [18] and present images generated during the model inversion process. Figure 9  
752 shows images generated from the prompt: "A female mannequin dressed in a black leather jacket and  
753 gold pleated skirt." and Figure 10 shows images generated from the prompt: "A big dog chasing a  
754 small kitten." In both figures, the first row displays results produced by our PMI+full-model inversion  
755 strategy, while the second row shows results generated by the baseline method from [18]. The label  
756 *steps* in each figure indicates the number of update steps during full-model inversion. Note that  
757 images from our method are initialized using PMI.

758 In both examples, our method generates meaningful images with fewer update steps, highlighting the  
759 effectiveness of the PMI+full-model inversion strategy. We note, however, that generating images  
760 from text does not directly reflect CL performance. This visualization is intended to illustrate that our  
761 method significantly reduces the number of update steps required during model inversion, while still  
762 capturing key semantic features efficiently.

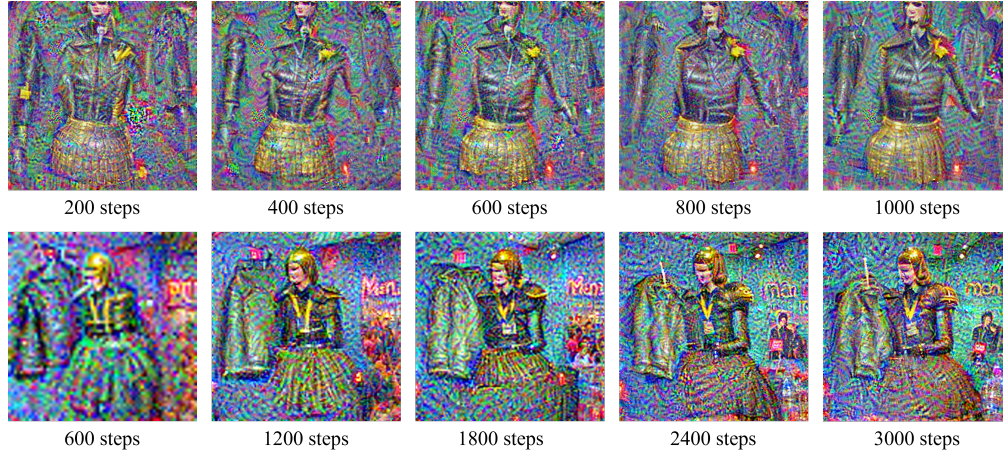


Figure 9: Images generated from the prompt "A female mannequin dressed in a black leather jacket and gold pleated skirt." during the model inversion process. The first row shows images generated using our PMI + full-model inversion strategy, while the second row presents images generated by the baseline method.

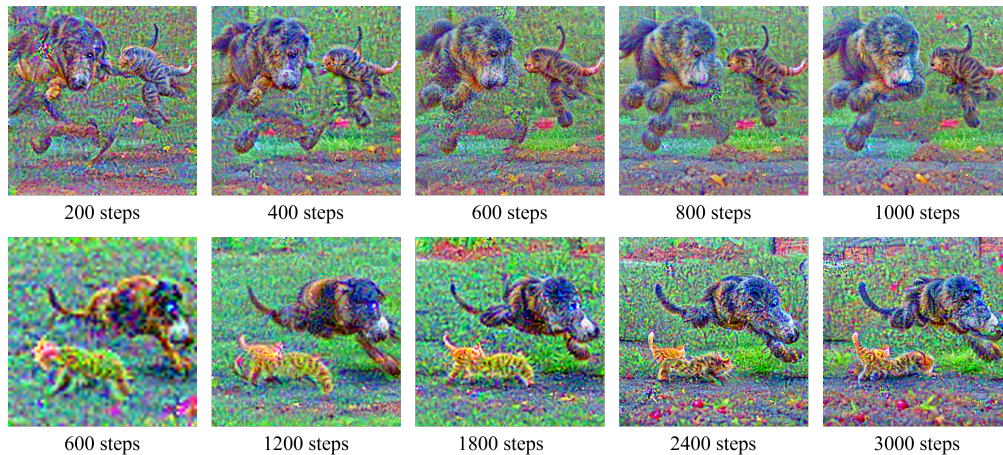


Figure 10: Images generated from the prompt "A big dog chasing a small kitten." during the model inversion process. The first row shows images produced using our PMI + full-model inversion strategy, while the second row displays images generated by the baseline method.

## I Experiment details

### I.1 ResNet-based CL

**Continual training.** The CIFAR-100 dataset contains 100 classes, which we evenly divide into 5, 10, and 20 disjoint tasks. Similarly, the Tiny-ImageNet dataset, consisting of 200 classes, is split into 5, 10, and 20 disjoint tasks. The class order for continual learning follows the setting used in R-DFCIL [10], and data augmentation includes random cropping and random horizontal flipping. The backbone model is ResNet-32 following the settings of R-DFCIL.

Each task takes totally 200 epochs for training, and learning rate is set to 0.1 with 0.5 times decreasing after every 40 epochs, we use SGD optimizer for all the experiments. Loss factors for hard knowledge distillation, relational knowledge distillation and classification head finetuning loss are set to 0.15, 0.5 and 1.0 respectively.

**Model inversion.** We maintain an image buffer containing 20,000 synthetic images, which are generated prior to training on each new task. The scaling factors for the MSE loss and the distribution constraint are set to 1.0 and 1.6, respectively. For model inversion, we use the Adam optimizer, with a learning rate of 0.8 for PMI and 0.15 for full-model inversion. The number of update steps is set to 100 for PMI and 280 for full-model inversion.

Table 9: Training hyper-parameters for CLIP-based CL, where LR denotes learning rate.

Method	CIFAR-100			ImageNet-R			CUB200		
	LR	Epoch	Optimizer	LR	Epoch	Optimizer	LR	Epoch	Optimizer
VPT	0.02	5	SGD	0.01	5	SGD	0.02	5	SGD
CODA-P	0.1	5	SGD	0.1	5	SGD	0.1	5	SGD
MoE-Adapter	0.001	3	AdamW	0.001	3	AdamW	0.001	10	AdamW

**Contrastive model.** The contrastive model is a two-layer MLP, with each layer consisting of 64 neurons and Leaky ReLU as the activation function. It is trained for 200 epochs using the SGD optimizer with a learning rate of 0.01. For CFS, we set the selection ratio to 0.5 and the number of selection steps to 40.

## I.2 CLIP-based CL

**Continual training.** In CLIP-based continual learning, we conduct experiments on CIFAR-100, ImageNet-R, and CUB-200. Both ImageNet-R and CUB-200 contain 200 classes, and we evenly split all three datasets into 10 disjoint tasks. Following the setting of PROOF [59], we use a fixed random seed (1993) to generate the class order. The optimization parameters are summarized in Table 9. For all methods, the loss weight for the hard knowledge distillation (HKD) loss is set to 0.1. For MoE-Adapter, we additionally use a loss weight of 0.2 for the text knowledge distillation loss and 0.001 for the text encoder fine-tuning loss on CIFAR-100 and ImageNet-R. For the fine-grained dataset CUB-200, we set both the text knowledge distillation and text encoder fine-tuning loss weights to 0.1.

**Model inversion.** We maintain 5 synthetic samples for each previously seen class. For model inversion, we use 200 update steps for PMI and 600 steps for full-model inversion. Following the setting in [18], we use the Adam optimizer and set the learning rate to 0.1 for both PMI and full-model inversion.

In experiments on CIFAR-100 and ImageNet-R, the loss weights for the MSE loss, distribution constraint, and smoothness constraint are set to 1.0,  $2 \times 10^{-3}$ , and  $5 \times 10^{-3}$ , respectively. For the fine-grained dataset CUB-200, the loss weights are set to 1.0 for the MSE loss, 0.2 for the distribution constraint, and  $5 \times 10^{-3}$  for the smoothness constraint.

**Contrastive model.** The contrastive model is implemented as a two-layer MLP, where each layer contains 512 neurons and uses Leaky ReLU as the activation function. It is trained for 200 epochs using the SGD optimizer with a learning rate of 0.01. For CFS, we use a selection ratio of 0.5 and perform 5 selection steps.

## I.3 Semantic aware feature projection

For both the CIFAR-100 and ImageNet-R datasets, we apply feature projection using the top 5 most similar classes, where similarity is measured by the cosine similarity between class text features. The parameter  $\alpha$  is set to 0.1 for both datasets.

## J Broader impact

In real-world applications, data availability is a major concern when deploying machine learning techniques. Model inversion addresses this issue effectively and is widely used in data-free scenarios, including data-free knowledge transfer, data-free meta-learning, and data-free continual learning. Additionally, model inversion can be employed to analyze what a model has learned, contributing to the development of trustworthy AI systems.

Model inversion generates data by extracting knowledge from a trained model. Applying model inversion to large pre-trained models allows for better utilization of the rich knowledge encoded in these models. Our work improves the efficiency of model inversion on large pre-trained CLIP models and demonstrates the potential for continually adapting models to new classes without requiring additional data collection—by recovering data directly from the pre-trained models. Furthermore, our method addresses the issue of feature distribution shift in model inversion-based continual learning, which can help reduce forgetting and improve overall performance.

822 We acknowledge that model inversion may be potentially use for recovering data containing unex-  
823 pected privacy appeared in training data, which could be a systematic issue of whole pipeline of data  
824 collection, training, model deployment, and inversion. However, our method is designed to improve  
825 the efficiency of model inversion and to better model class-wise feature distributions, rather than  
826 to recover private information. Moreover, model inversion relies on the knowledge encoded in the  
827 model; without privacy-related information being encoded, our method cannot recover any private  
828 data.

## 829 **K Computation resources and asset URLs**

830 All experiments are conducted on a system with four NVIDIA GeForce RTX 3090 GPUs (24 GB  
831 each) and an Intel(R) Core(TM) i9-12900K CPU with 64 GB of RAM.

832 Our experiments include the CIFAR-100, Tiny-ImageNet, ImageNet-R, and CUB-200 datasets. The  
833 URLs for these datasets are:

- 834 • <https://www.cs.toronto.edu/~kriz/cifar.html>
- 835 • <https://github.com/hendrycks/imagenet-r>
- 836 • [https://www.vision.caltech.edu/datasets/cub\\_200\\_2011/](https://www.vision.caltech.edu/datasets/cub_200_2011/)

837 Our ResNet-based continual learning experiments are implemented based on the open-source code of  
838 R-DFCIL [10] and DCMI [29]. The URLs for these implementations are:

- 839 • <https://github.com/jianzhangcs/R-DFCIL>
- 840 • [https://github.com/zihuanqiu/DCMI\\_CVPR24](https://github.com/zihuanqiu/DCMI_CVPR24)

841 Our CLIP-based continual learning experiments are implemented based on the open-source code of  
842 PROOF [59], CODA-Prompt [35], and MoE-Adapter [52]. The URLs for these implementations are:

- 843 • <https://github.com/LAMDA-CL/PROOF>
- 844 • <https://github.com/GT-RIPL/CODA-Prompt>
- 845 • <https://github.com/JiazuoYu/MoE-Adapters4CL>

846 The implementation of CLIP model inversion is based on [18]. The code URL is:

- 847 • <https://github.com/hamidkazemi22/CLIPInversion>

848 We confirm that all assets used in our work, including datasets, code, and pre-trained models, are  
849 used in accordance with their respective licenses.