Steering Prototypes with Prompt Tuning for Rehearsal-free Continual Learning

ICLR 2023 (under review)

Antnlp

Continual Learning

Objective:

$$\arg\min_{\Theta,\Phi} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}(g_{\phi^t}(f_{\theta^t}(\boldsymbol{x}_i^t)), y_i^t),$$

$\mathcal{T}_{1:T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_T\}$ A sequence of tasks $(x_i^t, y_i^t) \sim \mathcal{T}_t$

f_{θ^t} Feture extractor for task t (e.g. Bert with a prompt p_θ^t)

g_ϕ^t Task head for t (e.g. a classifier for text classification)

A Training-free baseline

Step 1: Calculate the prototype of class k by pre-trained model (e.g. Bert, ViT)

$$\mu_k = \frac{1}{|D_k|} \sum_{x \in D_k} f_\theta(x)$$

Note: freeze f_θ during training

Step 2: NEC (nearest-class-mean) classifier for classification

$$y^* = \arg \min_{y=1,\cdots,K} \{d(f_\theta(x), \mu_y)\}$$

Bad: Rely heavily on pre-trained models

Semantic drift and Prototype interference

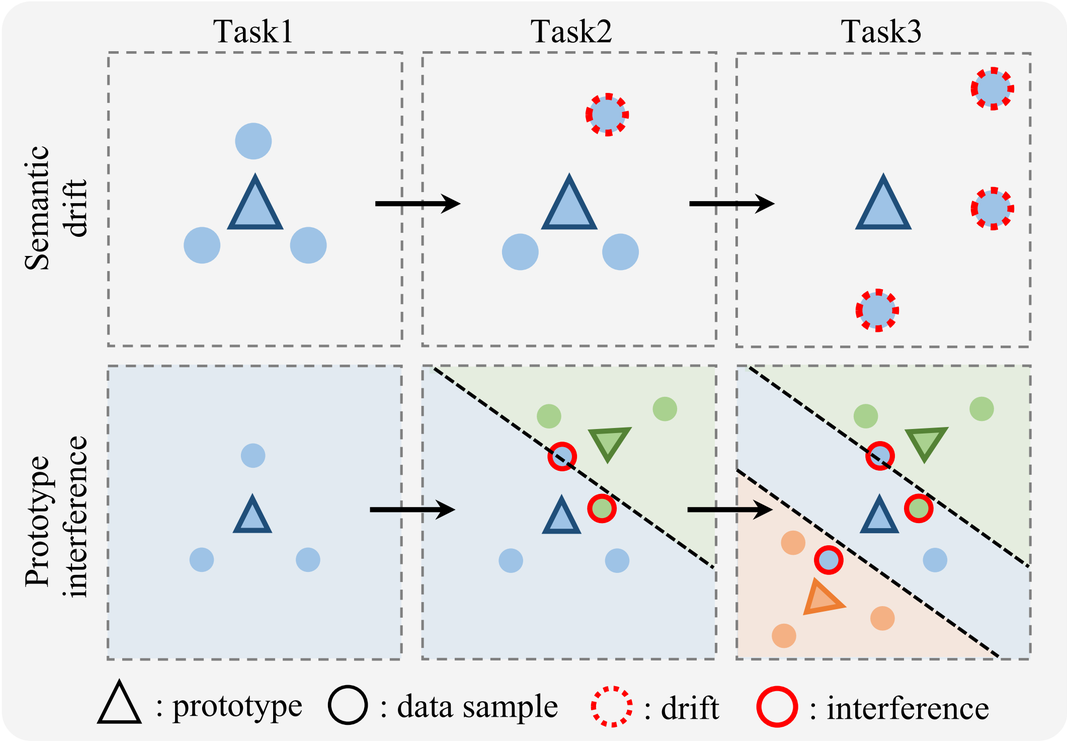
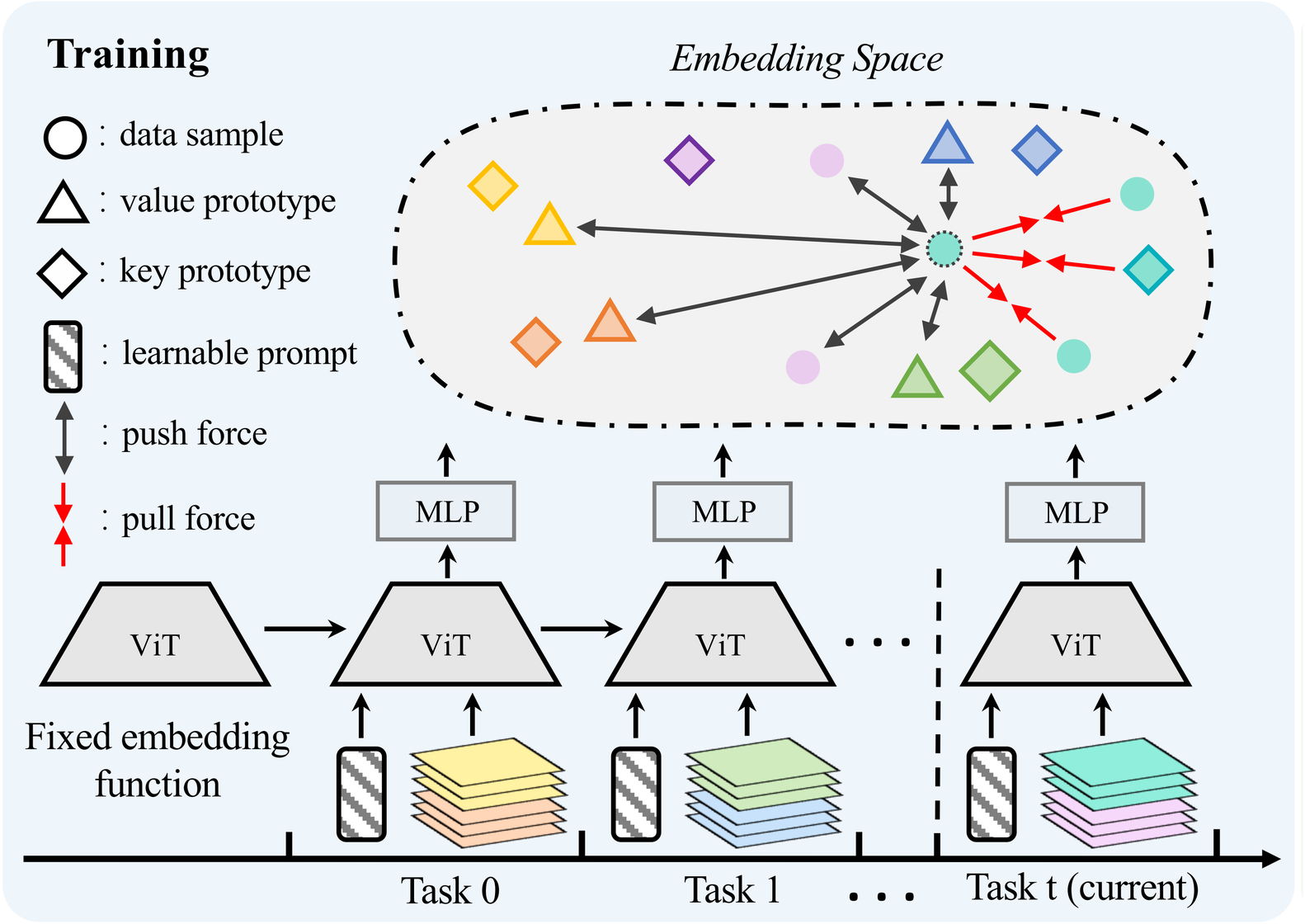


Figure 1: An illustration of semantic drift and prototype interference in the latent space. Both phenomena occur simultaneously in continual learning and cause *catastrophic forgetting*. Different colors represent different classes.

1. Semantic drift
2. Prototype interference

CPP - Contrastive Prototypical Prompt



$$\mu_k = \frac{1}{|D^k|} \sum_{x \in D^k} f_\theta(x)$$

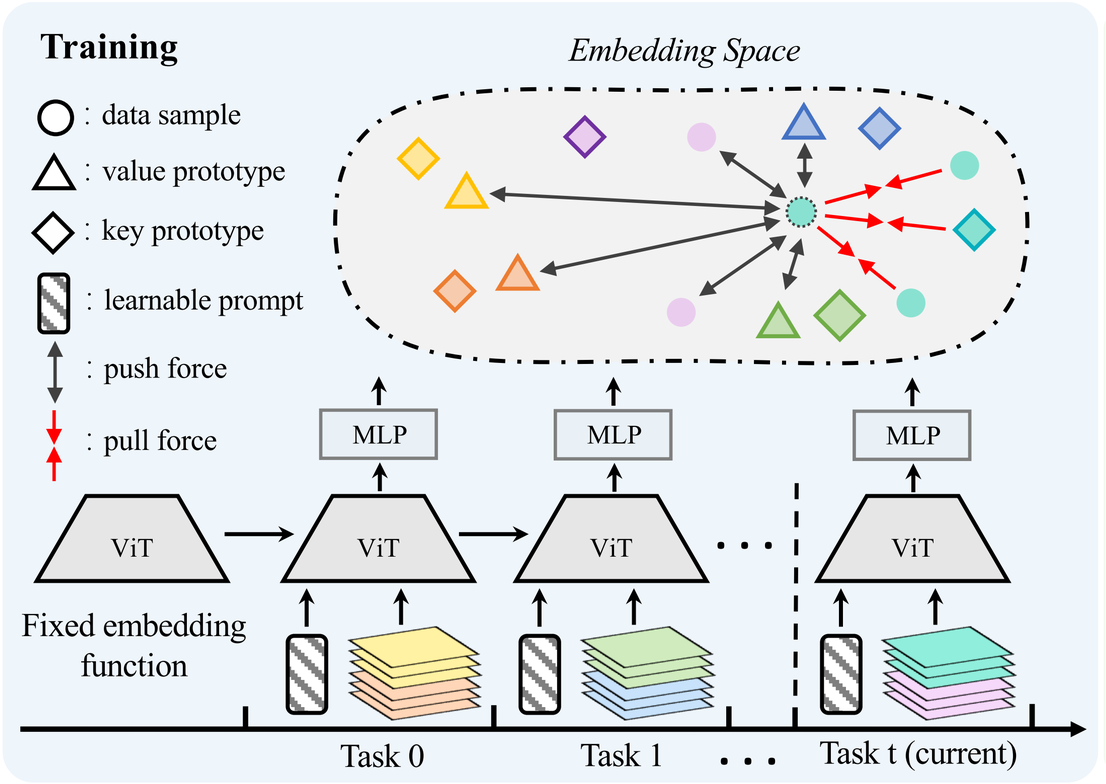
$$\mu'_k = \frac{1}{|D^k|} \sum_{x \in D^k} f_\theta(x; p^k)$$

$$z = m_{\sigma^t}(f_{\theta;P^t}(x))$$

CPP - Contrastive Prototypical Prompt

$$\mathcal{L}^k = \sum_{\boldsymbol{z}_i \in P(i)} \mathcal{L}_i^k = \sum_{\boldsymbol{z}_i \in P(i)} \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_p)/\tau)}{\sum_{\boldsymbol{z}_n \in N(i)} \exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_n)/\tau)},$$

Info NCE loss



$$P(i) = \{z_p \in Z : y_p = y_i = k\}$$

$$\hat{P}(i) = P(i) \cup \{\mu_k\}$$

$$N(i) = \{z_n \in Z : y_n \neq y_i\} \cup \{\mu'_1 \cdots \mu'_{k-1}\}$$

Prototype argumentation

$$\hat{\mu}_k = \mu_k + m * e, e \sim \mathcal{N}(0,1)$$

$$L_{task} = \frac{1}{M} \sum_{m=1}^{M} L^m$$

Training Algorithm

Algorithm 1: Training algorithm

Input: Pre-trained ViT model f_θ, number of tasks T, training epochs E, training set $\{(\boldsymbol{x}_i^t, y_i^t)\}_{i=1}^{n_t}\}_{t=1}^{T}$, prompt length L_p and centriod number C.

for $t = 1, \cdots, T$ **do**

  Initialize: MLP m_{σ^t}, prompt group P^t, $U = \varnothing$, $U' = \varnothing$

  for *class* $k \in \mathcal{Y}^t$ **do**

    Generate key prototype $\boldsymbol{\mu}_k$ with Eq. 2

    $U \leftarrow U \cup \boldsymbol{\mu}_k$

  end

  for $e = 1, \cdots, E$ **do**

    Optimize σ^t, P^t through generalized Eq. 6

  end

  Dispose m_{σ^t}

  $f_\theta \leftarrow f_{\theta, P^t}$

  for *class* $k \in \mathcal{Y}^t$ **do**

    Generate value prototype $\boldsymbol{\mu}'_k$ with Eq. 2

    $U' \leftarrow U' \cup \boldsymbol{\mu}'_k$

  end

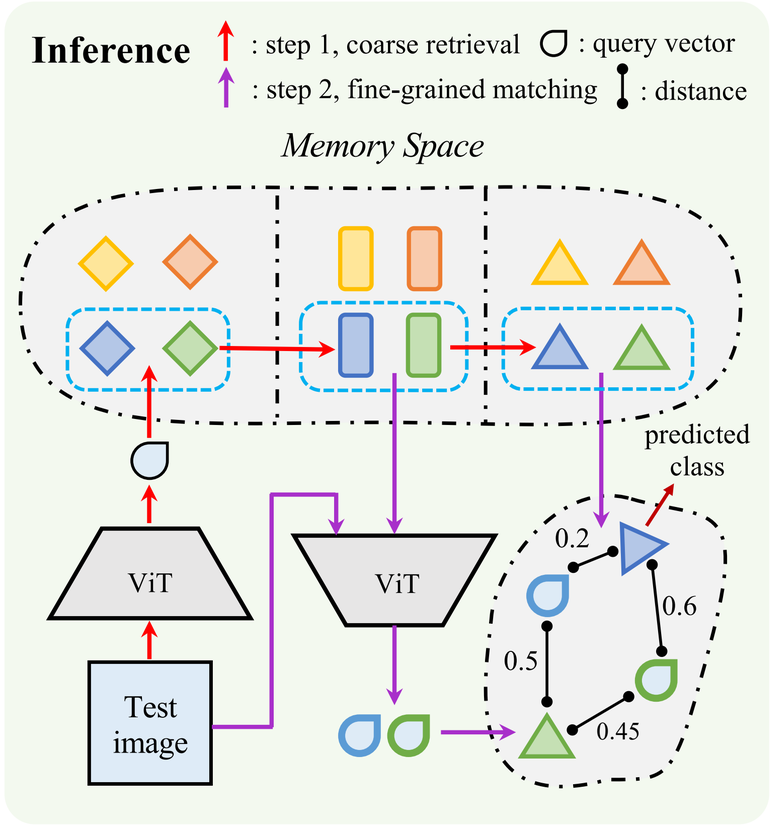
end

Output: $\{P^t\}_{t=1}^{T}$, U and U'.

$$\boldsymbol{\mu}_k = \frac{1}{|\mathcal{D}_k^t|} \sum_{\boldsymbol{x} \in \mathcal{D}_k^t} f_\theta(\boldsymbol{x}),$$

$$\mathcal{L}^k = \sum_{\boldsymbol{z}_i \in P(i)} \mathcal{L}_i^k = \sum_{\boldsymbol{z}_i \in P(i)} \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_p)/\tau)}{\sum_{\boldsymbol{z}_n \in N(i)} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_n)/\tau)},$$

CPP - Inference Stage



Algorithm 2: Inference algorithm

Given: Pre-trained ViT model f_θ, the collection of key prototypes U, the collection of value prototypes U', the collection of prompts $\{P^t\}_{t=1}^{T}$, query funcion $q(,,r)$ and pair-wise distance function d.

Input: test image $\boldsymbol{x}$

Initialize: $Q' = \varnothing, L = \varnothing$

$\boldsymbol{q} = f_\theta(\boldsymbol{x})[0,:]$; `// use class token as query vector`

$M = q(\boldsymbol{q}, U, r)$; `// retrieve indexes of r nearest key prototypes`

for $t \in M$ **do**

  $\boldsymbol{q}' = f_{\theta,P^t}(\boldsymbol{x})$

  $Q' \leftarrow Q' \cup \boldsymbol{q}'$

  $L \leftarrow L \cup \boldsymbol{\mu}'_t$

end

$y = \underset{y=1\ldots K}{\arg\min}(d(Q', L))$

Output: label y

Multi-centroids

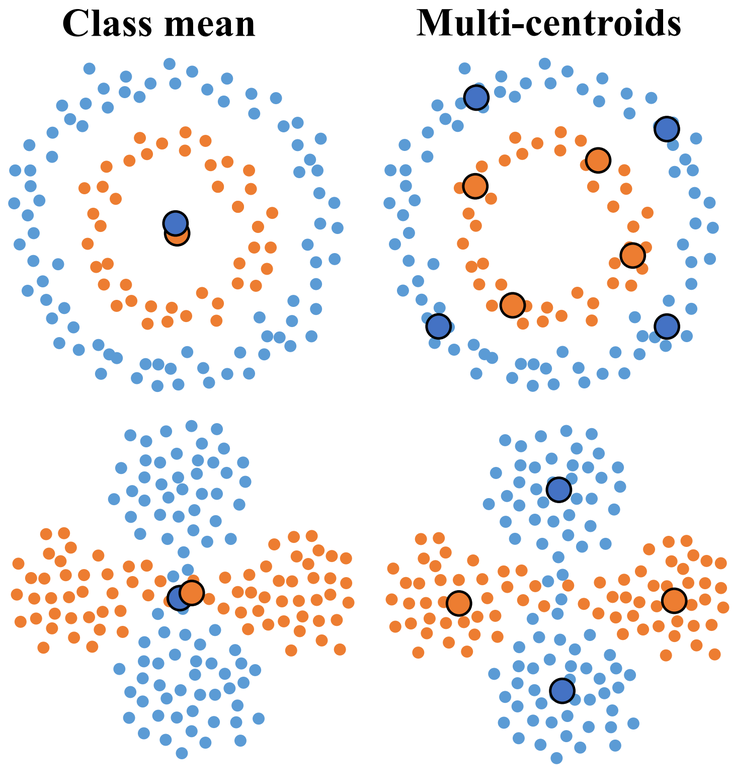


Figure 3: Two toy cases for average embedding prototype and multi-centroid prototype.

1. Spectral clustering[1]
2. kNN (*k*-nearest neighbors)

[1] On spectral clustering: Analysis and an algorithm.

Main Results

Table 1: Comparison with state-of-the-art rehearsal and rehearsal-free methods on split CIFAR-100 and 5-datasets. All results are reported using a ImageNet pre-trained ViT-B/16 for fairness.

Method	**Buffer size**	**Split CIFAR-100**		**Buffer size**	**5-datasets**	
		Avg. Acc (↑)	Forget (↓)		Avg. Acc (↑)	Forget (↓)
ER (Chaudhry et al., 2019b)	5000	82.53±0.17	16.46±0.25	500	84.26±0.84	12.85±0.62
BiC (Wu et al., 2019)		81.42±0.85	17.31±1.02		85.53±2.06	10.27±1.32
GDumb (Prabhu et al., 2020)		81.67±0.02	-		-	-
DER++ (Buzzega et al., 2020)		83.94±0.34	14.55±0.73		84.88±0.57	10.46±1.02
Co^{2}L (Cha et al., 2021)		82.49±0.89	17.48±1.80		86.05±1.03	12.28±1.44
FT-seq	0	33.61±0.85	86.87±0.20	0	20.12±0.42	94.63±0.68
EWC (Lopez-Paz & Ranzato, 2017)		47.01±0.29	33.27±1.17		50.93±0.09	34.94±0.07
LwF (Li & Hoiem, 2018)		60.69±0.63	27.77±2.17		47.91±0.33	38.01±0.28
L2P (Wang et al., 2022c)		83.86±0.28	7.35±0.38		81.14±0.93	4.64±0.52
DualPrompt (Wang et al., 2022b)		86.51±0.33	5.16±0.09		88.08±0.36	2.21±0.69
CPP (ours)		**89.43**± **0.24**	**3.61**±**0.31**		**93.36**±**0.03**	**0.1**±**0.01**
Upper-bound	-	90.85±0.12	-	-	93.93±0.18	-

Comparison

Table 2: Comparison with architecture-based methods on Split CIFAR-100. `Diff` (lower is better) measures how close the performance to the upper-bound of the used backbone.

Method	**Backbone**	**Avg. Acc (↑)**	**Diff (↓)**	**Buffer size**	**Additional Parameters** MB	%
Upper-bound	ResNet18	80.41	-	-	-	-
SupSup (Wortsman et al., 2020)		28.34±2.45	52.07	0	3.0	6.5%
DualNet (Pham et al., 2021)		40.14±1.64	40.27	1000	5.04	10.9%
RPSNet (Rajasegaran et al., 2019)		68.60	11.81	2000	181	404%
DynaER (Yan et al., 2021)		74.64	5.77	2000	19.8	43.8%
Upper-bound	ResNet152	88.54	-	-	-	-
DynaER (Yan et al., 2021)		71.01±0.58	17.53	2000	159	68.5%
Upper-bound	Customized ViT	76.12	-	-	-	-
DyTox (Douillard et al., 2022)		62.06±0.25	14.06	2000	0.04	0.38%
Upper-bound	ViT-B/16	90.85±0.12	-	-	-	-
L2P (Wang et al., 2022c)		83.86±0.28	6.99	0	1.94	0.56%
DualPrompt (Wang et al., 2022b)		86.51±0.33	4.34	0	1.90	0.55%
CPP (ours)		**89.43± 0.24**	**1.42**	0	**0.74**	**0.21%**

Comparison

Table 3: Comparison with prototype-related methods on split ImageNet-subset and split CIFAR-100.

Method	**Buffer size**	**Split CIFAR-100**		**Split ImageNet-subset**	
		Backbone	Avg. Acc (↑)	Backbone	Avg. Acc (↑)
Upper-bound	-	ViT	90.85±0.12	MAE	94.22±0.18
iCaRL	2000	ResNet18	51.12 ±0.36	ResNet18	23.77±0.35
ProtoAug	0	ResNet18	36.32±0.33	ResNet18	27.16±0.24
iCaRL	2000	ViT	75.10±0.26	MAE	87.96±0.26
ProtoAug	0	ViT	64.1±0.20	MAE	72.72±0.31
CPP (ours)	0	ViT	**89.43±0.24**	MAE	**93.90±0.12**

Ablation Study

Table 4: We ablate the proposed CPP and the multi-centroid prototypes with four different pre-training methods on split CIFAR-100. When both CPP and multi-centroid are not applied, the model is the training-free baseline model.

Pretrain	**CPP**	**Multi-centroids**	**Split CIFAR-100** Avg. Acc (↑)	Forgetting (↓)
Deit (Touvron et al., 2021)			71.9	9.97
	✓		80.32±0.6	8.36±0.74
		✓	74.6±0.18	8.42±0.13
	✓	✓	**81.33±0.37**	**6.28±0.53**
Dino (Caron et al., 2021)			76.69	8.91
	✓		80.82±0.22	6.16±0.09
		✓	79.71±0.09	7.72±0.04
	✓	✓	**83.73±0.14**	**4.87±0.06**
MAE (He et al., 2022)			74.65	8.6
	✓		80.26±0.46	8.74±0.25
		✓	76.71±0.17	8.21±0.05
	✓	✓	**82.28±0.38**	**6.65±0.33**
ViT (Dosovitskiy et al., 2021)			75.97	7.83
	✓		88.73±0.17	3.88±0.20
		✓	78.62±0.11	6.81±0.02
	✓	✓	**89.43± 0.24**	**3.61±0.31**

Training-free baseline

CPP

Ablation Study

SupCon

$$\mathcal{L}_{out}^{sup} = \sum_{i \in I} \mathcal{L}_{out,i}^{sup} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_p / \tau\right)}{\sum_{a \in A(i)} \exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_a / \tau\right)}$$

Table 5: Ablation study on contrastive prototypical loss and its alternatives.

Method	**Split CIFAR-100**	
	Avg. Acc (↑)	Forgetting (↓)
CE (w/o mlp)	37.12±2.54	10.01±1.63
CE	87.98±0.32	4.53±0.35
SupCon (w/o mlp)	48.03±6.97	7.37±2.42
SupCon	88.60±0.18	3.89±0.32
CPP (w/o mlp)	55.43±7.74	0.8±0.29
CPP (w/ uniformity)	88.82±0.18	4.01±0.15
CPP (w/o prototype)	88.85±0.20	3.88±0.27
CPP (w/o ProtoAug)	89.18±0.15	3.78±0.30
CPP	**89.43± 0.24**	**3.61±0.31**

MLP in Contrastive Learning (CV)

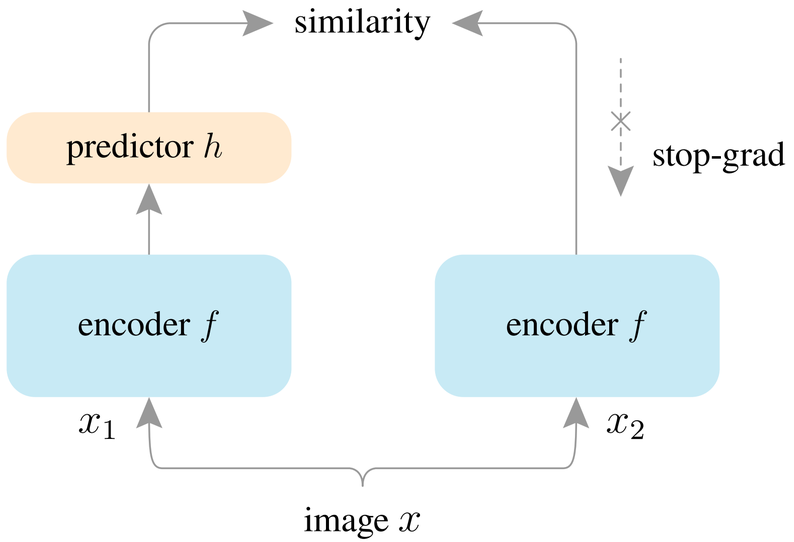
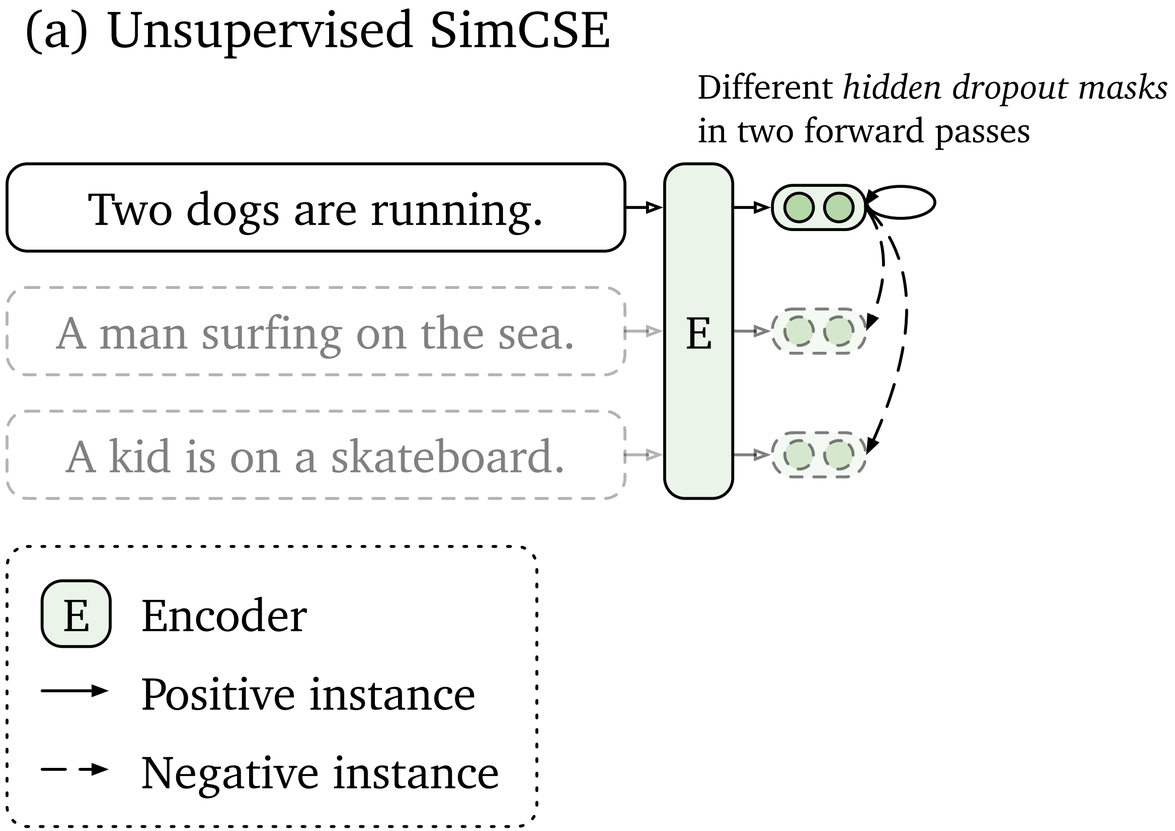


Figure 1. **SimSiam architecture**. Two augmented views of one image are processed by the same encoder network f (a backbone plus a projection MLP). Then a prediction MLP h is applied on one side, and a stop-gradient operation is applied on the other side. The model maximizes the similarity between both sides. It uses neither negative pairs nor a momentum encoder.

	pred. MLP h	acc. (%)
baseline	lr with cosine decay	67.7
(a)	no pred. MLP	0.1
(b)	fixed random init.	1.5
(c)	lr not decayed	68.1

Table 1. **Effect of prediction MLP** (ImageNet linear evaluation accuracy with 100-epoch pre-training). In all these variants, we use the same schedule for the encoder f (lr with cosine decay).

MLP in Contrastive Learning (NLP)



Pooler	Unsup.	Sup.
`[CLS]`		
w/ MLP	81.7	**86.2**
w/ MLP (train)	**82.5**	85.8
w/o MLP	80.9	**86.2**
First-last avg.	81.2	86.1

Table 6: Ablation studies of different pooling methods in unsupervised and supervised SimCSE. *`[CLS]` w/ MLP (train)*: using MLP on `[CLS]` during training but removing it during testing. The results are based on the development set of STS-B using $\text{BERT}_{\texttt{base}}$.

Ablation Study

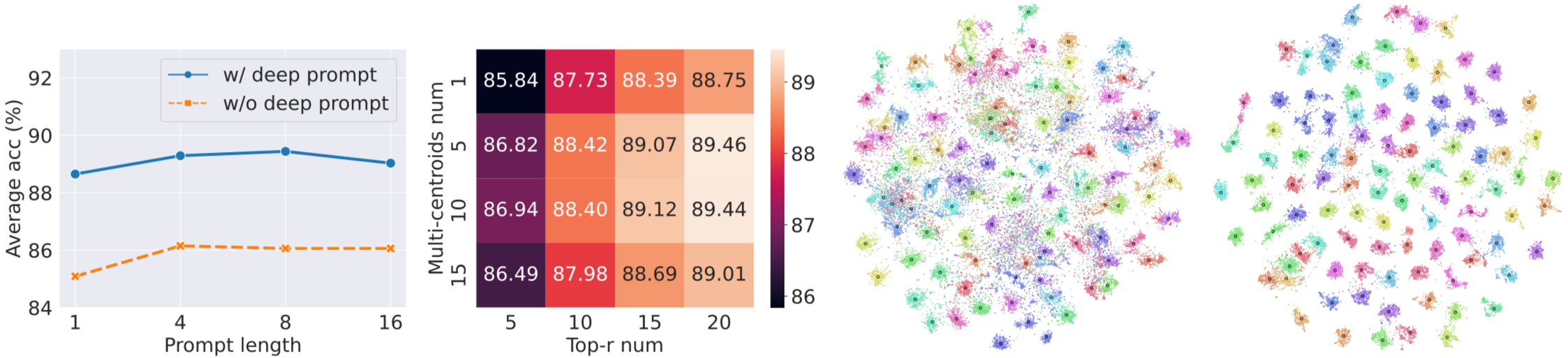


Figure 4: **Left:** ablation on prompt length and deep prompt. **Middle:** centroid number v.s. number of query neighbors. **Right:** t-SNE visualizations for samples w/ (right) and w/o (left) prompt groups.

Effectiveness of the query function

Effectiveness of the query function. We assume that samples with similar semantics should tend to locate close to each other in latent space. It is convincing to see that, giving r nearest neighbors, whether the true class falls in the candidates. In this case, top-r accuracy in the coarse query process can be deemed as a rigid upper-bound for CPP. We here plot top-r accuracy under the 5-centroid prototype environment in Fig. 6. We can see that top-r accuracy increases monotonically with r and $r = 20$ works fairly well. As such, query vector in combination with key prototypes can hence be leveraged to effectively retrieve candidate prompt groups.

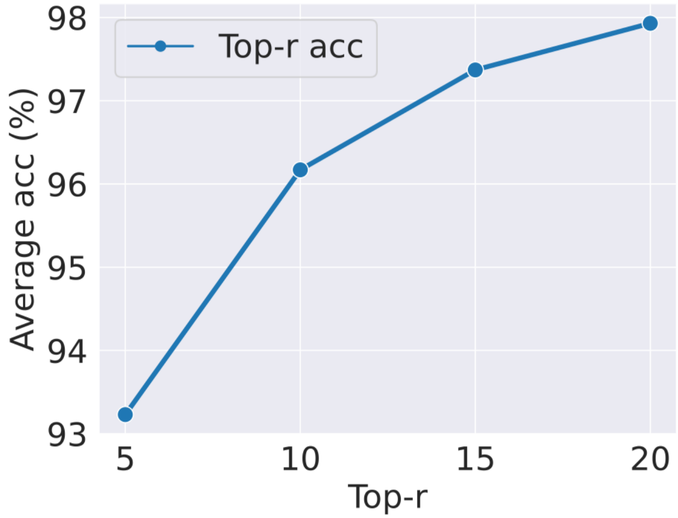


Figure 6: Top-r accuracy of split CIFAR-100 under 5-centroid prototype.

Table 7: Results on split cifar-100 under different MLP layer numbers.

Layer num	**Hidden units**	**Split CIFAR-100**	
		Avg. Acc (↑)	Forget (↓)
1	2048	82.16	4.58
3	1024	89.27	4.13
3	2048	**89.43**	3.61
3	4096	89.16	4.09
5	2048	88.54	**3.20**

Q&A