STEERING PROTOTYPES WITH PROMPT TUNING FOR REHEARSAL-FREE CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Prototype, as a representation of class embeddings, has been explored to reduce memory footprint or avoid bias towards the latest task for continual learning. However, prototype-based methods still suffer from performance deterioration due to semantic drift and prototype interference. In this work, we propose a simple and novel framework for rehearsal-free continual learning. We show that task-specific prompt-tuning when coupled with a contrastive loss design can effectively address both issues and largely improves the potency of prototypes. The proposed framework excels at three challenging benchmarks, resulting in 3% to 6% absolute improvements over state-of-the-art methods without usage of a rehearsal buffer or a test-time oracle. Furthermore, the proposed framework largely bridges the performance gap between incremental learning and offline joint learning, demonstrating a promising design schema for continual learning.

1 INTRODUCTION

Continual learning (Thrun, 1995), the capability of learning sequentially from a continuous stream of correlated data, is crucial for modern intelligent systems as the world is non-stationary (Hadsell et al., 2020). Yet, existing deep neural networks are known to be prone to *catastrophic forgetting* (McCloskey & Cohen, 1989): models suffer from dramatic performance degeneration on earlier learned tasks when learn new information. Prototype (i.e., the class mean embedding (Snell et al., 2017)) exhibits a promising functionality in continual learning context as it can retain previous knowledge in a data-efficient manner (Zhu et al., 2021) and avoid bias towards the latest task (Rebuffi et al., 2017) when coupled with a nearest class mean (NCM) (Mensink et al., 2013) classifier. However, prototypes themselves are also subject to abrupt efficacy drop due to *semantic drift* and *prototype interference*. Concretely, learning a sequence of tasks with a single model can be viewed as generating a sequence of snapshots of the model, and only the latest version is retained. Therefore, a data sample at inference and its corresponding prototype is, in fact, encoded by different embedding functions (except for data samples from the latest task). This inconsistency can cause severe drifts in latent space as shown in Fig. 1 (top). Besides, when new data samples that bear similar semantics with previous classes appear, their encoded features can locate near previous prototypes in latent space, thus causing interference as illustrated in Fig. 1 (bottom).

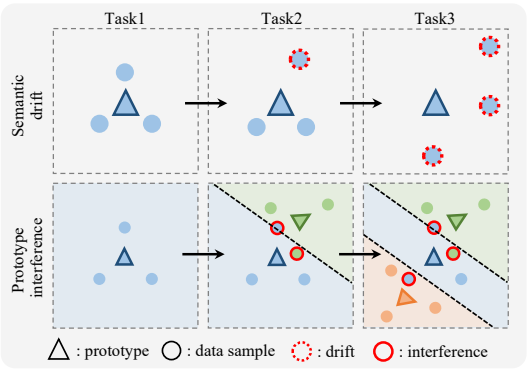


Figure 1: An illustration of semantic drift and prototype interference in the latent space. Both phenomena occur simultaneously in continual learning and cause *catastrophic forgetting*. Different colors represent different classes.

A recent transfer learning paradigm, namely prompt-tuning (Lester et al., 2021; Jia et al., 2022), demonstrates a strong knowledge adaption ability. It allows a tiny portion of extra learnable tokens to steer a frozen transformer-based architecture (Vaswani et al., 2017). Therefore, prompt-tuning reuses the pre-trained network in a parameter efficient manner without hurting its feature extraction

ability. Inspired by the efficiency of prompt-tuning and the plug-and-play property of the token, we propose a novel framework built upon the basis of task-specific prompt that can effectively address both semantic drift and prototype interference described above.

In our method, we associate the prototype of each class with a task-specific prompt group and maintain a collection of corresponding pairs in memory. During inference, we combine the task-specific prompt group with a frozen embedding function to reemerge each snapshot of the model. As such, we effectively eliminate the inconsistency between embedding functions used for prototypes generation and samples prediction. The frozen embedding function here can be deemed as consolidated global knowledge that keeps the system stable. Prompt groups, on the other hand, learn task-level specializations and maintain the plasticity of the system. To avoid prototype interference in embedding space, we train task-specific prompt groups with the designed contrastive prototypical loss. It encourages in-class clustering and increases inter-class distances giving a mixture of data embeddings and prototypes. Since we only maintain previous knowledge as prototypes and put them as anchors in latent space, the trained prompt groups can effectively steer prototypes to avoid interference without saving or replaying previous data samples. Furthermore, we propose the multi-centroid prototype strategy that leverages a group of fictitious embeddings instead of a mean embedding to characterize the distribution of a class in latent space. It helps to improve the representation power of prototypes and further mitigate semantic drift and prototype interference. The above schema effectively align both the space (i.e., the embedding space) and the embedding functions that are used during learning and inference, hence effectively boosting the potency of prototypes in continual learning.

We term our method **C**ontrastive **P**rototypical **P**rompt (CPP), a simple and novel continual learning framework that explores embedding space holistically. In experiments, CPP excels at split CIFAR-100, split ImageNet-subset and 5-datasets three challenging benchmarks, bringing around 3% to 6% absolute improvements over state-of-the-art methods. Moreover, it largely bridges the gap between incremental learning and offline joint learning[1]. The efficacy of proposed modules is thoroughly studied both empirically and analytically. The main contributions can be summarized as follows:

- We propose CPP, a simple and novel framework for rehearsal-free continual learning. It leverages contrastively learned task-specific prompt to effectively address both semantic drift and prototype interference issues.
- We present multi-centroid prototype strategy which can better characterize the class distribution and improves representativeness of prototypes. It is seamlessly merged into CPP and exhibits an additive benefit.
- CPP significantly outperforms the state-of-the-art methods and largely bridges the performance gap between incremental learning and offline joint-learning. The proposed modules are comprehensively analyzed and demonstrate clear and additive benefits.

2 Related Works

Continual learning. The development trajectory of continual learning is the history of combating against *catastrophic forgetting* (McCloskey & Cohen, 1989) issue. Existing algorithms can be mainly categorized into three subsets. Regularization-based methods (Lopez-Paz & Ranzato, 2017; Li & Hoiem, 2018) strike for a balance under *stability–plasticity dilemma*. They impose extra constraints on the changeability of network parameters while maintaining a certain degree of plasticity to learn new knowledge. Despite the succinct formulation, solely using regularization struggles when facing a long sequence of tasks (Hadsell et al., 2020). Architectural methods manage to overcome forgetting by allocating extra resources as learning progresses (Mallya & Lazebnik, 2018; Rusu et al., 2016; Pham et al., 2020). However, most existing methods assume the existence of a test-time oracle and face scalability issues. In practice, rehearsal-based methods (Buzzega et al., 2020; Cha et al., 2021) exhibit the most versatility and robustness through saving and rehearsing previous samples. Nevertheless, this strategy is sensitive to buffer size (Prabhu et al., 2020; Hadsell et al., 2020) and becomes infeasible under restricted scenarios (e.g., on edge devices, for privacy-sensitive applications). The proposed CPP here is a hybrid method. It combines merits from architectural and rehearsal-based methods without inheriting their limitations (see a full discussion in Appendix D).

[1]Joint training on all classes with i.i.d. assumption is deemed as the upper-bound for continual learning.

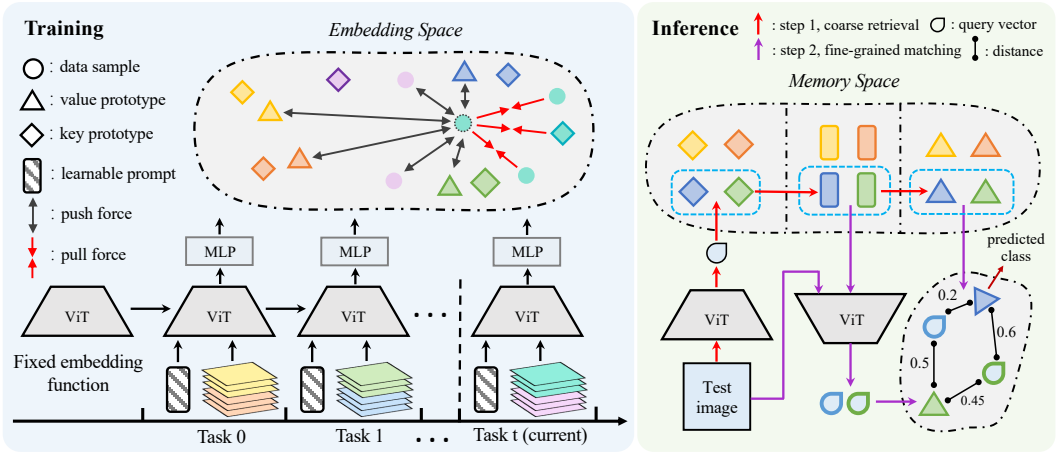


Figure 2: An overview of CPP. Different colors represent different classes. **Left:** along the learning process, knowledge from earlier tasks are retained as prototypes and are used as anchors in embedding space. Current prompt learn through avoiding interference. **Right:** during inference, a group of candidate prompt groups are first retrieved followed by a fine-grained matching process.

Prototypes for continual learning. It has been shown that embedding is less prone to information loss (Davari et al., 2022) and a typical linear classifier is one of the critical sources for abrupt forgetting due to the bias towards latest task (Zhang et al., 2021). As such, most prototype-related approaches (Rebuffi et al., 2017; Yu et al., 2020; Zhu et al., 2021) leverage prototypes in combination with a NCM classifier to discriminate data samples. Zhu et al. (2021), on the other hand, used prototypes as anchors in latent space to avoid semantic overlap and thus improving discrimination ability without forwarding explicit exemplars. Yu et al. (2020) managed to post-compensate semantic drifts of previous prototypes through approximating drifts from current data. Herein, instead of compensating drifts, CPP prevents drifts from the origin and handles prototype interference as well. Moreover, CPP deploys the multi-centroid prototype instead of a class mean embedding to better characterize the embedding distribution and improves representativeness of the prototype.

Prompt tuning. Initializing the model with pre-trained weights has become a *de facto* practice in both computer vision and natural language processing communities. However, a typical fine-tuning technique does not necessarily benefit when transferring models to downstream tasks (Kumar et al., 2022). Prompt-tuning (Li & Liang, 2021; Lester et al., 2021) has emerged as an alternative to reuse pre-trained knowledge. Jia et al. (2022) further adapted prompt-tuning to the vision domain. It has recently also been introduced to continual learning. Both L2P (Wang et al., 2022c) and DualPromt (Wang et al., 2022b) leveraged a prompt pool or global prompts that share across tasks to learn incremental knowledge. S-prompts (Wang et al., 2022a) used domain-specific prompts to tackle the domain-incremental learning. We here apply task-specific prompts to counteract semantic drifts and prototype interference, and leverage prototypes as classifiers without projecting to logistic space.

3 Methodology

In this section, we start with describing the problem setup and, along the way, introduce the notations (Sec. 3.1). Then we present a minimum feasible prototype-based framework which serves as a proof of concept and the baseline model (Sec. 3.2). Afterwards, We introduce the proposed CPP upon the baseline model (Sec. 3.3). At last, we describe multi-centroid prototype strategy (Sec. 3.4). Fig. 2 provides an overview of our framework.

3.1 Problem Setup and Notion

Supervised continual learning can be defined as learning a model over a sequence of T tasks $\mathcal{T}_{1:T} = \{\mathcal{T}_1, \mathcal{T}_2 ... \mathcal{T}_T\}$. Each task $\mathcal{T}_t$ is associated to a dataset $\mathcal{D}^t = \{(\boldsymbol{x}_i^t, y_i^t)_{i=1}^{n_t}\}$ containing n_t data pairs where $\boldsymbol{x}$ is the input vector and y is its corresponding label. Each data pair $(\boldsymbol{x}_i^t, y_i^t) \in (\mathcal{X}^t \times \mathcal{Y}^t)$

belongs to an unknown distribution $(\mathcal{X}^t \times \mathcal{Y}^t)$ and $\mathcal{Y}^t \cap \mathcal{Y}^{t'} = \emptyset$ while $t \neq t'$. Without loss of generality, a neural network at session t can be decoupled into an embedding function $f_{\theta^t}(\cdot) : \mathbb{R}^{W \times H \times C} \rightarrow \mathbb{R}^D$ and a classifier $g_{\phi^t}(\cdot) : \mathbb{R}^D \rightarrow \mathbb{R}^K$ that parameterized by θ^t and ϕ^t, respectively. Then the overall learning target is to minimize:

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

where $\mathcal{L}$ is a loss measurement, $\Theta = \{\theta^1 ... \theta^T\}$ and $\Phi = \{\phi^1 ... \phi^T\}$. Note that at each task $\mathcal{T}_t$, only dataset $\mathcal{D}^t$ is accessible. Most reigning methods assume an extra replay buffer to save samples from previous tasks and augment current dataset with the replay buffer. In the rehearsal-free setup (Wang et al., 2022c), we do not assume the existence of a replay buffer.

3.2 A Training-free Baseline

Let $\mathcal{D}_k^t$ denote a set of samples belonging to class k at session t, we compute a prototype for each class k as the mean embedding following Rebuffi et al. (2017):

$$\boldsymbol{\mu}_k = \frac{1}{|\mathcal{D}_k^t|} \sum_{\boldsymbol{x} \in \mathcal{D}_k^t} f_\theta(\boldsymbol{x}), \tag{2}$$

and save $\boldsymbol{\mu}_k$ to memory. θ is initialized by a pre-trained ViT (Dosovitskiy et al., 2021) and kept frozen across the whole process: $\theta^1 = \theta^2 = \cdots = \theta^T$. We maintain a collection of prototypes $U = \{\boldsymbol{u}_1, \boldsymbol{u}_2 ... \boldsymbol{u}_K\}$ for K classes that have been observed so far. Then we use the nearest-class-mean (NCM) (Mensink et al., 2013) classifier for classification:

$$y^* = \arg\min_{y=1...K} \{d(\boldsymbol{u}_y, f_\theta(\boldsymbol{x}))\}, \tag{3}$$

where $d : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ is a distance function measuring the distance between two D-dimensional embeddings. Here, we use the cosine distance following the common practice in self-supervised representation learning (Chen et al., 2020). This simple and training-free baseline produces promising results under a strong embedding function (see Table 4), confirming the crucial role played by the embedding and effectiveness of prototypes in the continual learning context.

3.3 Contrastive Prototypical Prompt

Steering prototypes with prompt. Ideally, a perfect static embedding function can project embeddings to the places that locate nearest to their corresponding prototypes in the latent space, thus preventing forgetting. However, in practice, an embedding function is ever-changing and samples from different categories yet with similar semantics can interleave in the latent space and cause interference. To this end, we leverage a group of extra learnable parameters (prompts) to adapt a fixed embedding function to up-to-now information and reemerge different snapshots of the model through combining it with different prompt groups. Specifically, we append a series of prompts $\boldsymbol{p}_i \in \mathbb{R}^{L_p \times D}$ to the existing tokens. L_p is the length of prompts, and D denotes the embedding dimensionality. The information flow of a transformer layer i is defined as:

$$[\boldsymbol{c}_i, \boldsymbol{e}_i] = T_i([\boldsymbol{c}_{i-1}, \boldsymbol{p}_{i-1}, \boldsymbol{e}_{i-1}]), \tag{4}$$

where T_i represents a multi-head self-attention block followed by a feed-forward block in the i^{th} layer. $\boldsymbol{c} \in \mathbb{R}^{1 \times D}$ denotes the class token and $\boldsymbol{e} \in \mathbb{R}^{L_e \times D}$ are existing tokens with length L_e. Operator $[\cdot]$ performs concatenation along the sequence length dimension. Here, we adopt deep prompt (Jia et al., 2022) by adding prompts to all S layers. The prompt group for a task t is denoted by $P^t = \{\boldsymbol{p}_1^t, \boldsymbol{p}_2^t ... \boldsymbol{p}_S^t\}$ and the embedding function can be rewritten as:

$$f_{\theta^t}(\cdot) \rightarrow f_{\{\theta, P^t\}}(\cdot). \tag{5}$$

We maintain a collection of prompt groups as learning progresses and each prompt group is associated with a group of key and value prototypes that will be illustrated later in this section.

Contrastive prototypical loss. To effectively learn the prompt group and leverage it reduce prototype interference, we use a contrastive formulation which explicitly encourages alignments between

embeddings and prototypes from the same class as well as pushing away embeddings and prototypes from different classes. In session t, let $I = \{(\boldsymbol{x}_1, y_1)...(\boldsymbol{x}_N, y_N)\}$ be a batch of N image pairs and $Z = \{\boldsymbol{z}_1...\boldsymbol{z}_N\}$ be their corresponding embeddings. We define $\boldsymbol{z} = m_{\sigma^t}(f_{\{\theta;P^t\}}(\boldsymbol{x}))$ where $m_{\sigma^t}(\cdot)$ is a multi-layer perception (MLP) parameterized by σ^t. Note that $m_{\sigma^t}(\cdot)$ is re-initialized at each new task and being disposed during inference. The learning objective for a target prototype (class) k is then defined as one-versus-all:

$$\mathcal{L}^k = \sum_{\boldsymbol{z}_i \in P(i)} \mathcal{L}_i^k = \sum_{\boldsymbol{z}_i \in P(i)} \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_p)/\tau)}{\sum_{\boldsymbol{z}_n \in N(i)} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_n)/\tau)}, \tag{6}$$

where $\text{sim}(\cdot, \cdot)$ denotes the similarity function and i is the index of a data sample with label k in the batch. $P(i) = \{\boldsymbol{z}_p \in Z : y_p = y_i = k\}$ is a set of positive samples w.r.t. image i and $\hat{P}(i) = P(i) \cup \{\boldsymbol{u}_k\}$ further includes the key prototype of class k; $N(i) = \{\boldsymbol{z}_n \in Z : y_n \neq y_i\} \cup \{\boldsymbol{\mu}'_1...\boldsymbol{\mu}'_{k-1}\}$ is a collection of negative samples with $\boldsymbol{u}'$ representing the value prototype of the previously learned classes. Eq. 6 can be naturally generalized to a task-wise formulation by averaging over all M classes within the current task: $\mathcal{L}_{task} = \frac{1}{M}\sum_{m=1}^{M} \mathcal{L}^m$. The embedding space in Fig. 2 illustrates the idea of the designed loss function. To better restrain the discrimination boundary, we further adopt prototype augmentation (Zhu et al., 2021) when using prototypes as negative anchors in denominator. Concretely, negative prototypes are randomly perturbed by a scaled Gaussian noise $\boldsymbol{e} \sim \mathcal{N}(0, 1)$ with same dimension: $\hat{\boldsymbol{\mu}}_k = \boldsymbol{\mu}_k + m * \boldsymbol{e}$, where scale factor m is calculated as the average variance of the corresponding class embeddings.

The proposed contrastive prototypical loss deviates from the canonical supervised contrastive loss (Khosla et al., 2020) in following aspects. 1) We add prototypes as positive and negative anchors to avoid prototype interference in latent space. For instance, new data sample can locate at a position in the latent space where it is preoccupied with other samples from previous classes. In this case, positive anchors can prevent the distribution from being over-squeezed and shifted, while negative anchors can retain spaces for previous data. 2) We only use a single view for each data sample, i.e., we do not transform a sample into multiple different views. 3) The designed loss function only focuses on alignments of positive embeddings and does not constrain the intra-class uniformity, which is considered as one of the pivot properties that attributes to the success of contrastive representation learning Wang & Isola (2020). Concretely, we do not pair samples from the same category as negative pairs in the denominator. Since NCM classifier discriminates by selecting the closest prototype, and increasing intra-class uniformity can enlarge the distance between a sample and its corresponding prototype which is against the classification policy. (see an analysis from the energy perspective in Appendix. B). We refer to Appendix A for an analysis of gradients.

Inference by reemerging model snapshots. To effectively reemerge each snapshot of the model, we decouple the prototype of a class into two-fold: a *key* prototype and a *value* prototype. The key prototype $\boldsymbol{\mu}$ is generated using the Eq. 2 at the beginning of a task. The value prototype $\boldsymbol{\mu}'$ is produced with Eq. 2 after inserting the learned prompt group by the end of the task. And we maintain a collection of key prototypes $U = \{\boldsymbol{u}_1, .., \boldsymbol{u}_k\}$ and value prototypes $U' = \{\boldsymbol{u}'_1, .., \boldsymbol{u}'_k\}$ along the learning process. During inference, a coarse query vector $\boldsymbol{q} : \mathbb{R}^{1\times D}$ is first generated followed by a query function $q(\boldsymbol{q}, U, r)$ to find r nearest key prototypes and retrieve their corresponding prompt groups $\{P^1...P^r\}$. Here, $\boldsymbol{q}$ is simply the class token from the last layer and the query function measures the pair-wise cosine similarity between $\boldsymbol{q}$ and key prototypes U. Then, we leverage retrieved prompt groups to generate a set of fine-grained queries $Q' = \{\boldsymbol{q}'_1...\boldsymbol{q}'_r\}$ where $\boldsymbol{q}'_r$ is the generated in the same way as $\boldsymbol{q}$ after inserting corresponding prompt group P^r. At last, the class of value prototype that poses the minimum distance among Q' will be the final prediction. Since the mismatched prompt group will increase distance between samples and their corresponding value prototypes and the correct prompt group will behave in an opposite way. Fig. 2 (right) depicts the information flow of the inference process. Please refer to Algs. 1 and 2 in Appendix C for summarization and see a discussion about inference efficiency in Appendix E.

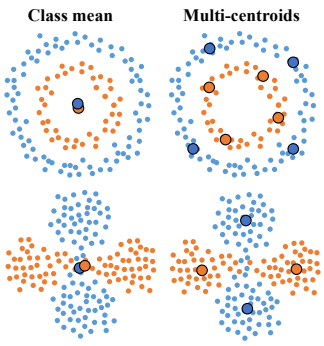


Figure 3: Two toy cases for average embedding prototype and multi-centroid prototype.

3.4 MULTI-CENTROID PROTOTYPES

Existing literature in continual learning simply adopts the mean embedding when it comes to prototypes (Yu et al., 2020; Zhu et al., 2021; Zhou et al., 2022). In this case, it implicitly assumes the distribution in the latent space to be convex (e.g., a Gaussian distribution), and the distance function belongs to Bregman divergence (Snell et al., 2017). This premise may not hold in practice as no strict constraints are imposed on embedding distributions, and cosine similarity is not one of Bregman divergences. Fig. 3 displays two toy cases where class mean embedding fails to be representative. To this end, we propose to multi-centroid prototypes. Instead of using mean embedding, we generate a group of fictitious embeddings to characterize the class distribution. Given a set of embeddings from class k, we first calculate similarity matrix $\boldsymbol{S}_k : \mathbb{R}^{N\times N}$ by measuring the pair-wise cosine similarity between all samples. We then perform spectral clustering (Ng et al., 2001) with $\boldsymbol{S}_k$ as affinity matrix to generate C centroids $\{\boldsymbol{u}_{k,c}\}_{c=1}^{C}$, where C is a hyper-parameter. To deploy this strategy, we substitute each prototype $\boldsymbol{u}_k$ to its corresponding multi-centroid prototype $\{\boldsymbol{u}_{k,c}\}_{c=1}^{C}$ (for both key and value prototypes) in its existence during both training and inference process.

4 EXPERIMENTS

4.1 DATASETS

Split CIFAR-100 is a commonly used benchmark in continual learning. Following the standard setup, we evenly split CIFAR-100 into 10 disjoint tasks. Existing literature also explores split CIFAR-100 under multiple different splits. As such, we also report detailed session-wise results under 5, 10, 20 splits in Appendix I.

5-datasets is a collection of CIFAR-10 (Krizhevsky, 2009), MNIST (Lecun et al., 1998), Fashion-MNIST (Xiao et al., 2017), SVHN (Netzer et al., 2011), and notMNIST (Bulatov, 2011). Each dataset containing 10 classes is treated as one learning task. 5-datasets serves as a fair analog of real-word scenarios where inter-task diversity is large.

Split ImageNet-subset is typically deemed as a challenging and scaled-up benchmark for continual learning. Following Douillard et al. (2022), we divide a subset (100 classes) of ImageNet (Deng et al., 2009) into 10 tasks with 10 classes per task.

4.2 CONFIGURATION AND EVALUATION METRIC

Configuration. We use the following dataset-agnostic configuration for all experiments if not state otherwise. We train CPP (initialized with ImageNet pre-trained ViT-B/16) for 50 epochs with a batch size of 256 using the AdamW optimizer (Loshchilov & Hutter, 2019). The initial learning rate is set to 1×10^{-3} and anneals to 1×10^{-6} according to the cosine scheduler. The prompt length L_p is set to 8, and we use deep prompt as default. The multi-centroid number C and the number of nearest

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	-

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. † reported from the original papers. ‡ reported in DualPrompt (Wang et al., 2022b)

Method	Backbone	Avg. Acc (↑)	Diff (↓)	Pretrained	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%**

neighbors r is set to 5 and 20, respectively. A 3-layer MLP with 2048 hidden units and 768 output dimension is randomly initialized at each session. We adopt transformations used in Dino (Caron et al., 2021) as our data augmentation, and all input images are resized to 224.

Evaluation metric. We report widely used average accuracy and forgetting from the end session (Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2019a; Wang et al., 2022b). All experiments run for 5 times with different seeds. We report the average and standard deviation for each metric. There are also a set of works reporting average accuracy across all sessions. As such, we provide detailed descriptions of evaluation metrics in Appendix H and results under both protocols in Appendix I.

4.3 Comparison with State of the Arts

Rehearsal and rehearsal-free methods. We compare CPP to representative regularization-based methods: *EWC* (Lopez-Paz & Ranzato, 2017), *LwF* (Li & Hoiem, 2018), advanced rehearsal-based methods: *ER* (Chaudhry et al., 2019b), *GDumb* (Prabhu et al., 2020), *BiC* (Wu et al., 2019), *DER++* (Buzzega et al., 2020), *Co^{2}L* (Cha et al., 2021), and state-of-the-art prompt-based methods: *L2P* (Wang et al., 2022c), *DualPrompt* (Wang et al., 2022b). We report results from Wang et al. (2022b) where all baseline methods are reproduced with a pre-trained ViT-B/16. *FT-seq* represents typical sequential fine-tuning with a single linear classifier. As shown in Table 1, despite the rehearsal-free property of regularization-based methods, their performances lag behind a lot. Rehearsal-based methods, on the other hand, produce decent results under large memory budget. Prompt-based methods achieve state-of-the-art performances without using a rehearsal buffer. Our method surpasses existing approaches by a large margin on split CIFAR-100 and 5-datasets in terms of both classification accuracy and forgetting.

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**

Architecture-based methods. It is non-trivial to migrate ConvNet-based architectural methods to transformer-based methods, so we adopt the metric from Wang et al. (2022b) to measure the difference between the method and its corresponding upper bound. Table 2 shows that CPP largely bridges the gap between incremental learning and joint learning on split CIFAR-100 dataset. Moreover, CPP outperforms other prompt-based methods using less than 50% of trainable parameters, leading to a

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 equivalent to the training-free baseline model.

Pretrain	CPP	Multi-centroids	Split CIFAR-100 Avg. Acc (↑)	Split CIFAR-100 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**

better memory efficiency which is one of the critical desiderata in continual learning (see Appendix F for a detailed analysis of scalability).

Prototype-related methods. Here, we compare our method with state-of-the-art prototype-based methods, ProtoAug (Zhu et al., 2021) and iCaRL (Rebuffi et al., 2017), on split CIFAR-100 and split ImageNet-subset. To be impartial and prevents information leakage, we reproduce both methods using a ImageNet pre-trained ViT-B/16, whereas supervised pre-training method is used for split CIFAR-100 and MAE pre-training method (self-supervised) is used for split ImageNet-subset. We then carefully tune hyper-parameters to avoid reckless fail (see Appendix G for details). As shown in Table 3, and in agreement with observations in Ramasesh et al. (2022), a pre-trained ViT backbone indeed significantly boost performances of existing methods. Nevertheless, CPP displays a cutting-edge performance under the same backbone, manifesting a systematic advantage of our method over the existing prototype-based methods.

4.4 Ablation Study

Effectiveness of proposed modules. Since embeddings are one of the key ingredients in our recipe, it is crucial to analyze CPP upon different embedding functions. To this end, we implement CPP on four up-to-date pre-training methods, *ViT* (Dosovitskiy et al., 2021), *Deit* (Touvron et al., 2021), *Dino* (Caron et al., 2021) and *MAE* (He et al., 2022) that sweep supervised and self/un-supervised learning as well as discriminative and generative models. As displayed in Table 4, both proposed modules are robust w.r.t. all four pre-training methods, bringing around 10% absolute improvements over the baseline models. Each design remains effective when being isolated, and the benefits are additive when combined. An interesting observation is that different pre-training methods can cause large performance variances from the prototype perspective and there is a positive correlation ($\rho = 0.60$) between performances of the baseline models and final results. We deem this as an informative discover that leaves further probe in future work.

Contrastive prototypical loss outperforms alternatives. In our framework, the designed asymmetric contrastive loss explicitly aligns the optimization target with the classification problem, but it is still critical to validate the design empirically. As such, we first compare our designed loss with two widely-used alternatives: *CE* (cross-entropy) and *SupCon* (supervised contrastive loss) (Khosla et al., 2020). Then we independently add uniformity (w/ uniformity), remove prototypes (w/o prototype) and cancel prototype augmentation (w/o ProtoAug) to show the efficacy of each proposed component. As shown in Table 5, the proposed loss consistently outperforms other loss functions by a clear margin. Among different alternatives, *SupCon* is the most compatible, demonstrating the benefits of unifying optimization and classification space. In agreement with our intuition and analysis in Appendix B, encouraging uniformity results in a clear drop in performance, and removing prototypes

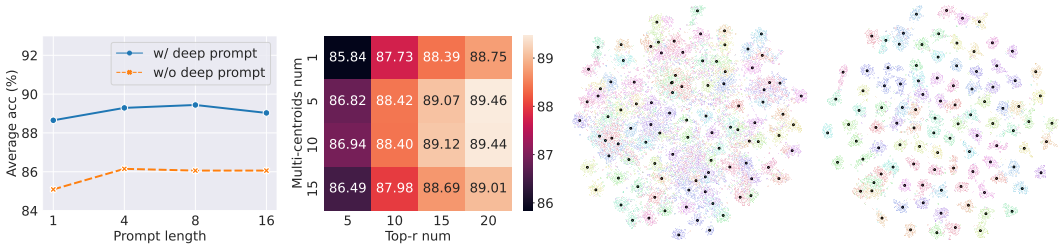


Figure 4: **Left:** ablation on prompt length and deep prompt. **Middle:** centroid number v.s. number of query neighbors. **Right:** t-SNE visualizations for samples w/ (right) and w/o (left) prompt groups.

(both positive and negative anchors) leads to inferior space allocation in the latent space. In addition, using prototype augmentation can also boost the performance.

MLP is non-negligible. We show in Table 5 that non-linearity introduced by the MLP is vital to the success of training prompt groups regardless of the loss design. This result coincides with the conventional practice in self-supervised representation learning, where MLP consistently improves the quality of representations.

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**

Ablation for prompts. Two factors in prompt design can affect the final performance: prompt length (which indicates the number of trainable units at each layer) and deep prompt (which represents adding prompts to all layers instead of the first layer). As shown in Fig. 4 (left), deep prompt consistently outperforms the shallow prompt, suggesting the importance of steering features at different levels of abstraction. Also, an appropriate length can improve the performance. It is worth pointing out that CPP can still outperform existing methods by a large margin even with $L_p = 1$ (using less than $1/20$ of parameters compared with DualPrompt). This result showcases a great parameter efficiency of our method which is critical towards the real-world scalable continual learning.

Centroid number v.s. query radius. Both centroid number C and query radius r can impact how many prompt groups are actually retrieved for fine-grained matching. For example, with a fixed r, increasing C may result in fewer categories being visited and vice versa. Even though one can always traverse all prompt groups to avoid querying process, it will increase inference time as the task accumulates. As such, it is more cost-effective to select a proper combination of C and r. Fig. 4 (middle) exhibits the result of a simple grid search on CIFAR-100, and the searched setting ($C = 5$ and $r = 20$) works fairly well for all other datasets.

Visualizations. We visualize data samples and their corresponding prototypes (single centroid) from CIFAR-100 with and without inserting learned prompt groups. As displayed in Fig. 4 (right), while samples from the same class tend to locate near each other in the latent space, samples from different classes still interleave with each other. After prompt groups are added, samples from the same category are tightly clustered, while different classes are spread out. See Appendix K for additional visualizations and analysis.

5 Conclusion

In this study, we propose a simple and novel framework for rehearsal-free continual learning. It leverages task-specific prompt to reemerge each snapshot of a model so as to avoid semantic drift. It also uses prompt-tuning to steer prototypes to reduce interference in the latent space through contrastive learning on the mixture of data embeddings and prototypes. Empirically, CPP surpasses state-of-the-art methods by a large margin without using a rehearsal buffer or a test-time oracle. We comprehensively analyze the effectiveness of proposed components, showcasing clear and additive benefits. We believe CPP can shine a light on the design principle of real-world continual learning giving current advances in architecture design and representation learning.

REFERENCES

Yaroslav Bulatov. Notmnist dataset. *Google (Books/OCR), Tech. Rep.[Online]. Available: http://yaroslavvb. blogspot. it/2011/09/notmnist-dataset. html*, 2, 2011.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and SIMONE CALDERARA. Dark experience for general continual learning: a strong, simple baseline. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 15920–15930. Curran Associates, Inc., 2020.

Mathilde Caron, Hugo Touvron, Ishan Misra, Herv'e J'egou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9630–9640, 2021.

H. Cha, J. Lee, and J. Shin. Co2l: Contrastive continual learning. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9496–9505, 2021.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*, 2019a.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019b.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

MohammadReza Davari, Nader Asadi, Sudhir Mudur, Rahaf Aljundi, and Eugene Belilovsky. Probing representation forgetting in supervised and unsupervised continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16712–16721, June 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. Dytox: Transformers for continual learning with dynamic token expansion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Raia Hadsell, Dushyant Rao, Andrei Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 24:1028–1040, 12 2020.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16000–16009, June 2022.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *European Conference on Computer Vision (ECCV)*, 2022.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 18661–18673. Curran Associates, Inc., 2020.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations*, 2022.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Yann LeCun, Sumit Chopra, Raia Hadsell, Fu Jie Huang, and et al. A tutorial on energy-based learning. In *PREDICTING STRUCTURED DATA*. MIT Press, 2006.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, November 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597. Association for Computational Linguistics, aug 2021.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pp. 109–165. Academic Press, 1989.

Thomas Mensink, Jakob J. Verbeek, Florent Perronnin, and Gabriela Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:2624–2637, 2013.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pp. 849–856. MIT Press, 2001.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. ISSN 0893-6080.

Quang Pham, Chenghao Liu, Doyen Sahoo, and HOI Steven. Contextual transformation networks for online continual learning. In *International Conference on Learning Representations*, 2020.

Quang Pham, Chenghao Liu, and Steven Hoi. Dualnet: Continual learning, fast and slow. In *Advances in Neural Information Processing Systems*, volume 34, pp. 16131–16144, 2021.

Ameya Prabhu, Philip Torr, and Puneet Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *The European Conference on Computer Vision (ECCV)*, August 2020.

Jathushan Rajasegaran, Munawar Hayat, Salman Khan, Fahad Shahbaz Khan, and Ling Shao. Random path selection for incremental learning. *Advances in Neural Information Processing Systems*, 2019.

Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations*, 2022.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, G. Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5533–5542, 2017.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

S. Thrun. A lifelong learning perspective for mobile robot control. In V. Graefe (ed.), *Intelligent Robots and Systems*. Elsevier, 1995.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 18–24 Jul 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 9929–9939, 2020.

Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning, 2022a.

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer G. Dy, and Tomas Pfister. Dualprompt: Complementary prompting for rehearsal-free continual learning. 2022b.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022c.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In *Advances in Neural Information Processing Systems*, volume 33, pp. 15173–15184. Curran Associates, Inc., 2020.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *CVPR*, pp. 374–382, 2019.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. 2021.

Lu Yu, Bartlomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6980–6989, 2020.

Chi Zhang, Nan Song, Guosheng Lin, Yun Zheng, Pan Pan, and Yinghui Xu. Few-shot incremental learning with continually evolved classifiers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021.

Da-Wei Zhou, Fu-Yun Wang, Han-Jia Ye, Liang Ma, Shiliang Pu, and De-Chuan Zhan. Forward compatible few-shot class-incremental learning, 2022.

Fei Zhu, Xu-Yao Zhang, Chuang Wang, Fei Yin, and Cheng-Lin Liu. Prototype augmentation and self-supervision for incremental learning. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

A Detailed Derivations for Contrastive Prototypical Loss

Here, we provide an analysis of gradients for proposed contrastive prototypical loss. It is sufficient to show gradients for a single prototype k. To ease the notion, we abbreviate similarity between vector $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ as $s_{i,j}$. Therefore, the loss of a sample $\boldsymbol{x}_i$ w.r.t. prototype k is:

$$\mathcal{L}_i^k = \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \log \frac{\exp(s_{i,p}/\tau)}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)} \tag{7}$$

The gradient with respect to the similarity $s_{i,j}$ between a positive pair $(\boldsymbol{z}_i, \boldsymbol{z}_j)$ where $j \in P(i)$ can be derived as:

$$\begin{aligned}
\frac{\partial \mathcal{L}_i^k}{\partial s_{i,j}} &= \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \frac{\partial}{\partial s_{i,j}} \left(s_{i,p}/\tau - \log \sum_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau) \right) \\
&= \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \left(\frac{1}{\tau} \cdot \mathbb{1}[p=j] - \frac{\frac{\partial}{\partial s_{i,j}} \left(\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau) \right)}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)} \right) \\
&= \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \left(\frac{1}{\tau} \cdot \mathbb{1}[p=j] - 0 \right) \\
&= \frac{1}{\tau |\hat{P}(i)|}
\end{aligned} \tag{8}$$

Here $\mathbb{1}$ is an indicator. Similarly, the gradient with respect to the similarity $s_{i,m}$ between a negative pair $(\boldsymbol{z}_i, \boldsymbol{z}_m)$ where $m \in N(i)$ is:

$$\begin{aligned}
\frac{\partial \mathcal{L}_i^k}{\partial s_{i,m}} &= \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \frac{\partial}{\partial s_{i,m}} \left(s_{i,p}/\tau - \log \sum_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau) \right) \\
&= \frac{-1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \left(0 - \frac{\frac{\partial}{\partial s_{i,m}} \left(\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau) \right)}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)} \right) \\
&= \frac{1}{|\hat{P}(i)|} \sum_{\boldsymbol{z}_p \in \hat{P}(i)} \left(\frac{\exp(s_{i,n}/\tau) \cdot \frac{1}{\tau} \cdot \mathbb{1}[n=m]}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)} \right) \\
&= \frac{1}{\tau |\hat{P}(i)|} \frac{\exp(s_{i,m}/\tau)}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)}
\end{aligned} \tag{9}$$

Taking gradients of loss $\mathcal{L}^k$ with respect to the similarity between a positive pair $s_{i,j} = \text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)$ and a negative pair $s_{i,m} = \text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_m)$ result in:

$$\frac{\partial \mathcal{L}_i^k}{\partial s_{i,j}} = \frac{1}{\tau |\hat{P}(i)|}, \qquad \frac{\partial \mathcal{L}_i^k}{\partial s_{i,m}} = \frac{1}{\tau |\hat{P}(i)|} \cdot \frac{\exp(s_{i,m}/\tau)}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)}. \tag{10}$$

The above derivation shows that positive similarities are treated equally and scaled by the temperature and the cardinality of the set of positive anchors. And property of implicit hard-case mining (i.e., proportional to the exponential term $\exp(s_{i,m}/\tau)$) is inherited from a typical contrastive loss in negative term.

B CPP IS AN ENERGY-BASED MODEL

The overall objective of an energy-based model (LeCun et al., 2006) (EBM) is to obtain an energy function $E_\theta(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$ parameterized by θ that maps the high dimensional input $\boldsymbol{x}$ to a scalar value. Giving an energy function $E_\theta(\cdot)$, probability density $p(\boldsymbol{x})$ can be expressed through Gibbs distribution:

$$p(y|\boldsymbol{x}) = \frac{\exp(-E_\theta(\boldsymbol{x}, y)/\tau)}{\int_{y'} \exp(-E_\theta(\boldsymbol{x}, y')/\tau)} = \frac{\exp(-E_\theta(\boldsymbol{x}, y)/\tau)}{\exp(-E_\theta(\boldsymbol{x})/\tau)} \tag{11}$$

where $E_\theta(x)$ is the ***Helmholtz free energy*** and τ is the temperature factor. As such:

$$E_\theta(x) = \tau \cdot -\log \int_{y'} \exp(-E_\theta(\boldsymbol{x}, y')/\tau) \tag{12}$$

When making final prediction under our framework, the categorical distribution can be represented as:

$$p(y|\boldsymbol{x}) = \frac{\exp(s_{x,y}/\tau)}{\sum_{y'=1}^{K} \exp(s_{x,y'}/\tau)} \tag{13}$$

where $s_{x,y} = \text{sim}(f_\theta(\boldsymbol{x}), \boldsymbol{\mu}_y)$. Note that we here merge the prompt parameters P into θ for the sake of simplicity. When connecting Eq. 13 with Eq. 11 and let $E_\theta(\boldsymbol{x}, y) = -s_{x,y}$, we see that the energy of $\boldsymbol{x}$ can be expressed as:

$$E_\theta(\boldsymbol{x}) = \tau \cdot -\log \sum_{y=1}^{K} \exp(s_{x,y}/\tau) \tag{14}$$

which is dominated by the largest similarity $s_{x,y*}$ given an appropriate temperature τ. Above analysis drives to the conclusion that predicting the class of prototype which is most similar to a given query vector will generate lowest energy for the system (i.e., a more stable system). Now the question turns to whether the proposed contrastive prototypical loss serves as a qualified energy loss function.

To see this, we first simplify the Eq. 7 to a formulation where there is only one positive sample $\boldsymbol{z}_{\hat{p}}$:

$$\begin{aligned} \mathcal{L}_i^k &= -\log \frac{\exp(s_{i,\hat{p}})}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)} \\ &= -s_{i,\hat{p}} + \log \sum_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau) \end{aligned} \tag{15}$$

when letting $\boldsymbol{z}_{\hat{p}}$ to be the value prototype $\boldsymbol{\mu}'_k$ that used as classifier, we have:

$$\mathcal{L}_i^k = \underbrace{-\text{sim}(\boldsymbol{z}_i, \boldsymbol{\mu}'_k)}_{\text{push down energy for prototype k}} + \underbrace{\log \sum_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau)}_{\text{pull up energies for other prototypes}} \tag{16}$$

As shown above, to minimize above loss, the first term will push down the energy for value prototype $\boldsymbol{\mu}'_k$ and the second term will increase energies for other prototypes. So above simplified loss is an effective loss function for the energy model. However, the *ground-truth* prototype $\boldsymbol{\mu}'_k$ is unavailable at training, instead a rough approximation $\boldsymbol{\mu}_k$ can be generated with Eq. 2 and a set of data samples are available. As such, Eq. 7 treat $\boldsymbol{\mu}_k$ and every sample embedding as positive prototypes and pulling the $\boldsymbol{z}_i$ to all of them simultaneously. This is equivalent to pulling $\boldsymbol{z}_i$ to a fictitious prototype that dynamically evolves with the distribution of embeddings. Since $\boldsymbol{\mu}'_k$ is generated using the learned embeddings at the end of the task, Eq. 7 still approximately minimizes the energy between a sample and its correspondingly value prototype $\boldsymbol{\mu}'_k$ even though $\boldsymbol{\mu}'_k$ is not explicitly shows in loss function.

Finally, we show that encouraging uniformity is against the principle of the energy model. By encouraging uniformity as typical supervised or self-supervised contrastive loss (Chen et al., 2020;

Khosla et al., 2020), we turn Eq. 15 into:

$$\begin{aligned}\mathcal{L}_i^k &= -\log \frac{\exp(s_{i,\hat{p}})}{\sum\limits_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau) + \sum\limits_{\boldsymbol{z}_p \in \hat{P}(i)} \exp(s_{i,p}/\tau)} \\ &= -s_{i,\hat{p}} + \log \left(\sum_{\boldsymbol{z}_n \in N(i)} \exp(s_{i,n}/\tau) + \underbrace{\sum_{\boldsymbol{z}_p \in \hat{P}(i)} \exp(s_{i,p}/\tau)}_{\text{harmfully pull up energy for target prototype}} \right)\end{aligned} \tag{17}$$

As shown above, we can see that the second term in $\log$ function acts adversely with respect to $-s_{i,\hat{p}}$ which minimizes the energy between a sample and its corresponding prototype. Hence, we intentionally remove the term that encourages the uniformity in typical contrastive loss from our designed loss.

During inference, the prediction process can be interpreted as selecting a prompt group that generates the most compatible embedding $\boldsymbol{z}'$ that has minimum energy with respect to a local system and its nearest value prototype is the predicted class. The Local system is generated through querying and making visible a predefined number of neighbors on the key manifold (i.e., manifold containing key prototypes).

C ALGORITHMS FOR CPP

The pipeline of the proposed framework is summarized in Algoritm 1 and Algorithm 2.

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, ··· , T do
    Initialize: MLP m_{σ^t}, prompt group P^t, U = ∅, U' = ∅
    for class k ∈ Y^t do
        Generate key prototype μ_k with Eq. 2
        U ← U ∪ μ_k
    end
    for e = 1, ··· , E do
        Optimize σ^t, P^t through generalized Eq. 6
    end
    Dispose m_{σ^t}
    f_θ ← f_{θ,P^t}
    for class k ∈ Y^t do
        Generate value prototype μ'_k with Eq. 2
        U' ← U' ∪ μ'_k
    end
end
```

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

D RELATIONS WITH PREVIOUS METHODS

There exists different taxonomies for continual learning methods (Parisi et al., 2019; Hadsell et al., 2020), we here take notions from Hadsell et al. (2020). Put conclusion first, CPP in this study is a hybrid method. From the view of prompt deployment, CPP is in consistent with modular models. Extra capacity is assigned when encountering new tasks and a specialization at task-level is maintained. Nevertheless, CPP does not necessarily suffer from computational issues and the

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

premise of test-time oracle as other modular methods. Inheriting parameter efficiency from prompt-tuning (Lester et al., 2021), CPP introduces negligible extra parameters for each incremental task. And since only prompts are updated with gradient descent and each task is associate with a fixed amount of prompts, the computational overhead is relatively small and constant. During inference, we leverage prototypes as key values and embeddings as quries to retrieve candiate prompt groups and thus avoid the requirement of test-time oracle. Taking prototype perspective, CPP can be categorized as a memory-based method, especially episodic memory method. We save prototypes in memory space and leverage to retain previous knowledge and also as classifiers during inference. Yet, unlike most memory-based method, we maintain information in a highly abstract and compressed manner and set them as anchors in latent space without forwarding them through the network.

E Discussion of Inference Efficiency

Here, we analyze the efficiency of inference process for CPP and provides an engineering solution. The time complexity of different data samples can be different during inference and there is randomness. For example, when querying 5 nearest neighbors with key prototypes, it does not necessarily result in 5 different prompt groups due the existences of multi-centroid prototypes and task-level prompt groups. At worst case, when 5 nearest centroids are from totally different classes and these classes are contained in totally different tasks. Then the query function will return 5 different prompt groups. However, in practice, centroids from same class tend to locate near to each other and a prompt group is shared by all classes within a task. It results in much lesser prompt groups that being retrieved and number of prompt groups vary according to data samples. From an engineering perspective, one can leverage batch processing to accelerate the process. Concretely, one can directly append all prompt groups and input a batch of attention masks to differentiate different configurations.

F Discussion of scalability

The scalability of a continual learning framework is one of the most crucial considerations in practice. It requires a framework to be first, memory efficient, consuming affordable memory footprint as tasks accumulate; second, computational efficient, using as less computational resources as possible; at last, privacy respectful, making it compatible with diverse real-world scenarios. We manage to analysis the scalability of our method with respect to these three aspects in the following paragraph.

For memory usage, there are two parts in our framework will cause increasing parameters during continual learning, prototypes and prompt groups. Using split CIFAR-100 as an instance, each new class will introduce $2 \times M \times 768$ extra parameters where M denotes the centroid number and the factor 2 is due to the decoupling of key and value prototypes. Let $M = 5$ as in our setting, we only save 10×768 extra parameters for a new class, this consumes approximately only $1/20$ of memory as saving a single ImageNet image ($224 \times 224 \times 3$). Besides, each new task (containing 10 classes)

will bring a prompt group (w/ deep prompt) with $8 \times 12 \times 768$ parameters. When averaged over 10 classes, it is approximately equivalent to 10×768 per class. Note that the increasing rate of parameters introduced by prompt group are negatively correlated to the size of the task. When put together both sources of extra parameters, we can see that for each incremental class we have roughly 20×768 parameters which costs 0.01536 MB and is equivalent to $1/10$ of a single ImageNet image. Moreover, the increment of memory usage for each class is constant w.r.t. all scenarios from where saving explicit samples may suffer from memory surge due to the resolution change (e.g., with a 4K camera). With above discussion, we believe it is fair to say that our framework is benign to scalability issue in terms of memory usage. From a computational perspective, thanks to prompt-tuning, only a tiny portion of parameters are updated through backpropagation. And prototypes are leveraged as anchors in latent space, thus no explicit data samples from previous classes need to be forwarded through the network. So the computational cost during the training is also minimized. Finally, since all information from previous tasks are retained as a few latent vectors (i.e., the prototypes), the privacy is inherently protected.

G Reproduction Details

Prototype-related methods. In Sec. 4.3, we compare our method to other representative prototype-based methods. To be impartial, we first run the original codes (ResNet-18 as feature extractor) on the same split ImageNet-subset and split CIFAR-100 as we used. In original setting of ProtoAug (Zhu et al., 2021), it uses 50 classes in initial session and 5 class for each incremental session. To be consistent with our setup, we change it to 10 classes per session and 10 sessions in total. Both iCaRL (Rebuffi et al., 2017) and ProtoAug (Zhu et al., 2021)'s technical designs are orthogonal to the choice of feature extractor. So we replace ResNet-18 with a pre-trained ViT-B/16 without the loss of fairness and keep other designs the same as originals. To take advantages of the pre-trained backbone, we set learning rate to 1e-4 for both methods according to a simple grid search and use the same training configuration as detailed in Sec. 4.2 for fairness.

DualPrompt on split ImangeNet-subset. Here, we further reproduce DualPrompt (Wang et al., 2022b) on split ImageNet-subset. The result can be seen in Table 6. To prevent information leakage, we use MAE pre-trained weights instead of the original ViT pre-trained weights. All other parameters are set following the original paper. Specifically, we set $L_e = 20, L_g = 5, start_e = 3, end_e = 5, start_g = 1, end_g = 2$. We train the model for 50 epochs with constant learning rate 0.005 and Adam optimizer is used. Since the original paper does not use split ImageNet-subset, there may exist a better configuration with further tuning.

Table 6: Reproduction of DualPrompt on split ImageNet-subset.

Methods	**split ImageNet-subset**	
	Avg. Acc (↑)	Forget (↓)
DualPrompt (MAE)	92.5	2.0
CPP (ours)	**93.9**	**1.89**

H Evaluation Metrics

Let $A_{i,j}$ be classification accuracy on the j-th task after training on the i-th task. After the model finishes training on the i-th task, we compute the Average Accuracy (A_i) and Forgetting (F_i) as follows:

$$A_i = \frac{1}{i} \sum_{j=1}^{i} A_{i,j}$$

$$F_i = \frac{1}{i-1} \sum_{j=1}^{i-1} \max_{j' \in \{1,\cdots,i-1\}} (A_{j',j} - A_{i,j})$$

Assume there are T tasks in total, we report accuracy from last session as $Acc = A_T$ following (Lopez-Paz & Ranzato, 2017; Wang et al., 2022b). There are also a large body of literature (Li &

Hoiem, 2018; Zhu et al., 2021; Douillard et al., 2022) report macro average over all sessions as $Acc = \frac{1}{T}\sum_{i=1}^{T} A_i$. To ease future reference, we provide results under both protocols in Appendix I.

I Results under Different Protocols

Detailed results for CIFAR-100 under different splits. Here, we provide session-wise results for split CIFAR-100 under different splits. As shown in Fig. 5, our method exhibits a clear and consistent improvements over other methods and the gap is enlarged as the length of task sequence increases.

Results under different metrics. Here, we provide results under two commonly used measurements as described in Appendix H.

Table 7: Results for CPP under different metrics.

Task num	Dataset	Pre-train	Accuracy		Forgetting	
			Avg. (↑)	Last (↑)	Avg. (↓)	Last (↓)
5	split CIFAR-100	ViT	92.6	89.58	3.99	3.97
10	split CIFAR-100	ViT	93.06	89.43	3.11	3.61
20	split CIFAR-100	ViT	92.49	88.25	3.66	4.56
5	5-datasets	ViT	95.15	93.36	0.12	0.1
10	split ImageNet-Sub	MAE	95.01	93.90	0.84	1.89

J Extra Ablations

Ablation for MLP design. As MLP layer is crucial for training prompts, we are curious about relations between MLP width (number of hidden units), deepth (layer numbers) and prompt quality. As shown in Table 8, either monotonously increasing layers or hidden units do not necessarily bring benefits. And 3-layer with 2048 hidden units, which is the same as the conventional practice in self-supervised representation learning, produces best performance in our framework. So we adopt this setting as default for all our experiments.

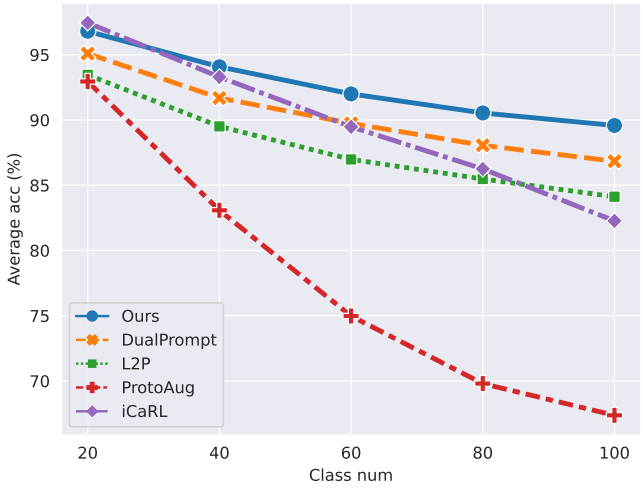
Table 8: Results on split cifar-100 under different MLP layer numbers.

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**

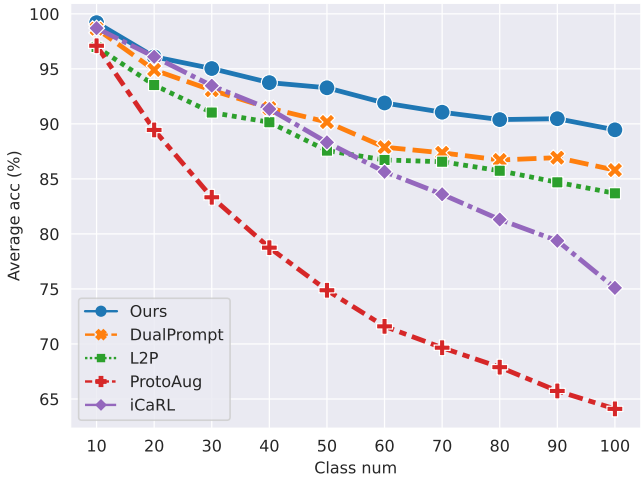
Generation of multi-centriod prototypes. In CPP, we leverage spectral clustering to generate multi-centroid prototypes. Herein, we also provide results for commonly used k-means clustering algorithm. As shown in Table 9, spectral clustering empirically demonstrates better performance and we thus take it as default.

Table 9: Different clustering algorithms for generating multi-centroid prototypes.

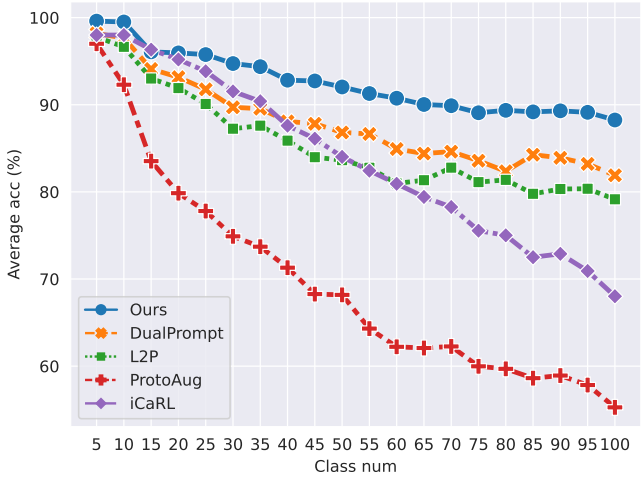
Methods	Split CIFAR-100	
	Avg. Acc (↑)	Forget (↓)
K-means	89.02	4.18
Spectral Clustering	**89.43**	**3.61**



(a) 5-splits.



(b) 10-splits.



(c) 20-splits.

Figure 5: Comparison with state-of-the-art methods on CIFAR-100 under multiple splits.

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. Hence, query vector in combination with key prototypes and a reasonable hyper-parameter can be safely leveraged to retrieve candidate prompt groups.

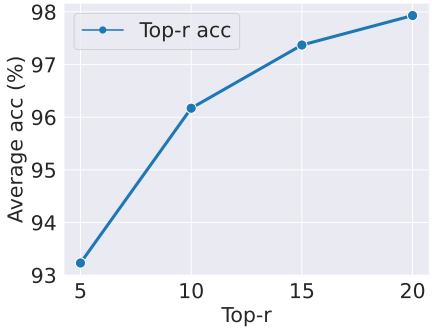
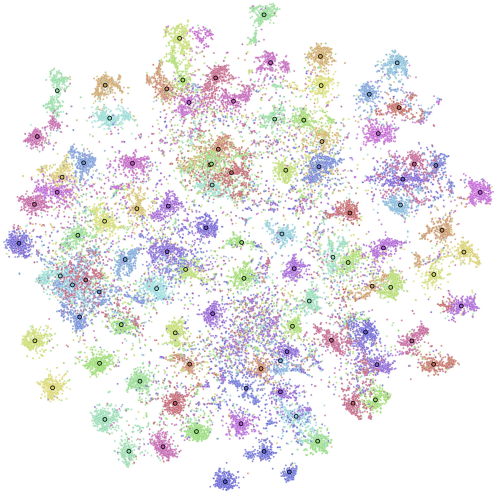


Figure 6: Top-r accuracy of split CIFAR-100 under 5-centroid prototype.

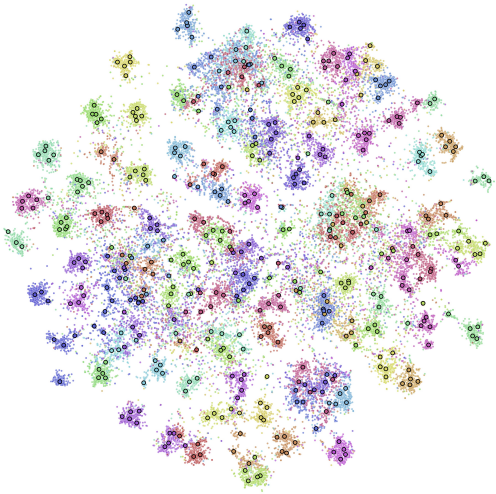
K Detailed Visualizations and Analysis

Fig. 7 displays training samples from CIFAR-100 in latent space under different configurations. Fig. 7a and Fig. 7b shows original data samples with their corresponding key prototypes and multi-centroid key prototypes, respectively. As shown in figures, both single-centroid and multi-centroid key prototypes effectively characterize the distribution for each class. In Fig. 7c and Fig. 7d, when replacing key prototypes to value prototypes, there is a clear drift and mismatch between class distributions and their corresponding prototypes. Since value prototypes characterize the distribution of prompted samples in latent space, this observation manifests a clear distribution shift in latent space when adding prompts. And thus justify the necessity of decoupling prototypes into the key prototypes and value prototypes two sets. Fig. 7e and Fig. 7f shows value prototypes and embeddings of samples after adding prompts. Both single-centroid and multi-centroid value prototypes suits the learned distributions well according to visualizations, while multi-centroid value prototypes can better capture outliers and thus being more representative.

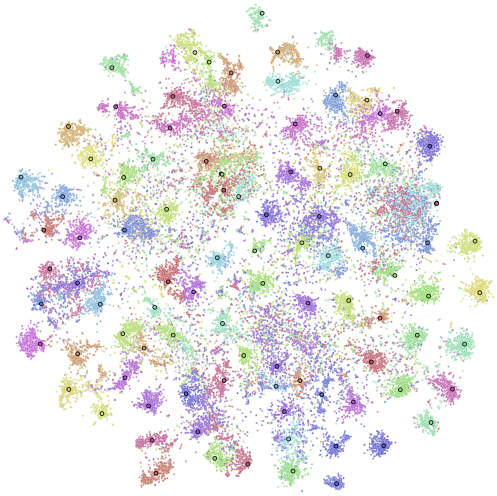
In Fig. 7, we have successfully shown the efficacy of key and value prototypes for representing training embeddings. So we test their performances on test dataset in Fig. 8, When leveraging key prototypes for coarse retrieval, it works fairly well according to Fig. 8a and Fig. 8b. Fig. 8c and Fig. 8d further validate the necessity of decoupling prototypes from test data view. There are some classes where key prototypes can still effectively characterize sample distribution after inserting prompts, suggesting less semantic overlap (easy to discriminate) and minor distribution shift. However, most classes fail to reuse key prototypes. When using value prototypes as classifiers for final prediction, Fig. 8e and Fig. 8f demonstrate a clear match which in turn results in high accuracy.



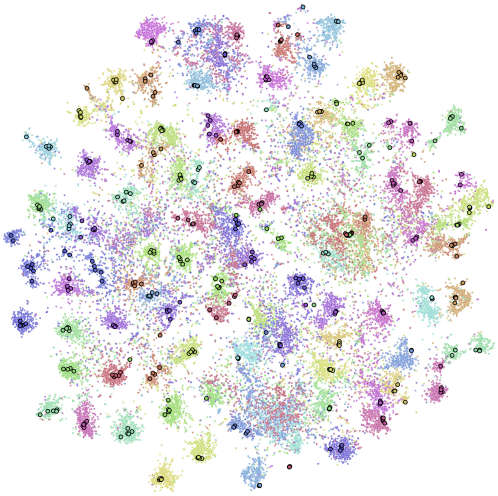
(a) Original train samples with key prototypes



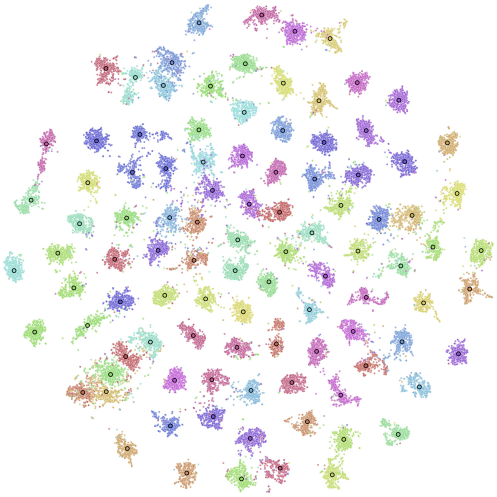
(b) Original train samples with multi-centriod key prototypes



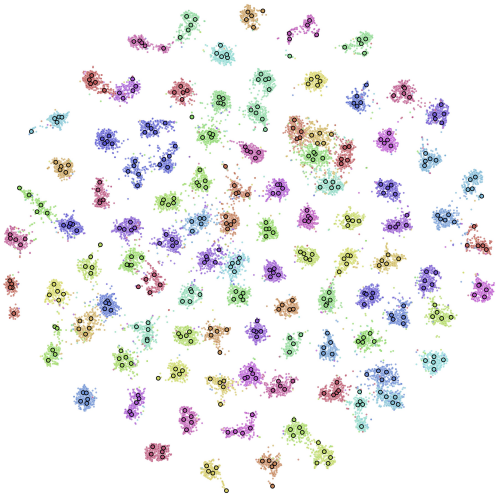
(c) Original train samples with value prototypes



(d) Original train samples with multi-centroid value prototypes

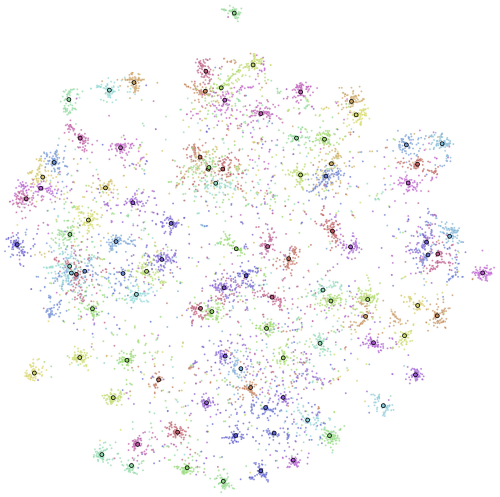


(e) Prompted train samples with value prototypes

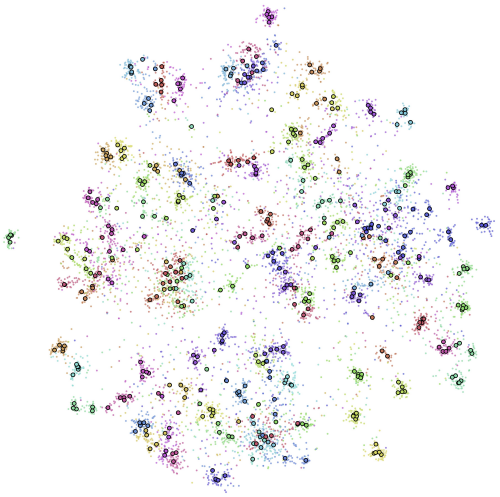


(f) Prompted train samples with multi-centriod value prototypes

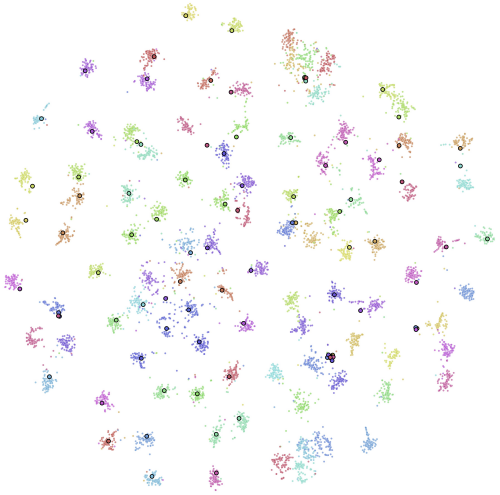
Figure 7: Visualization for train data in CIFAR-100.



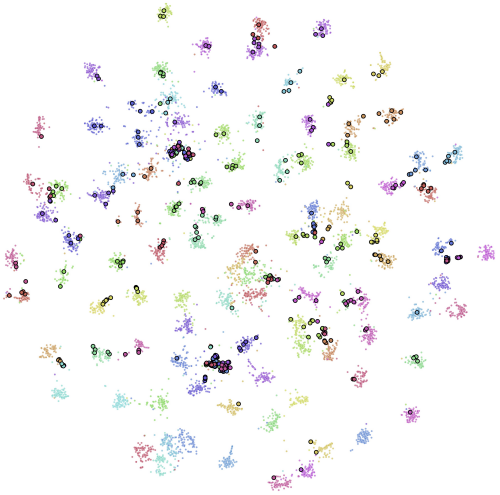
(a) Original test samples with key prototypes



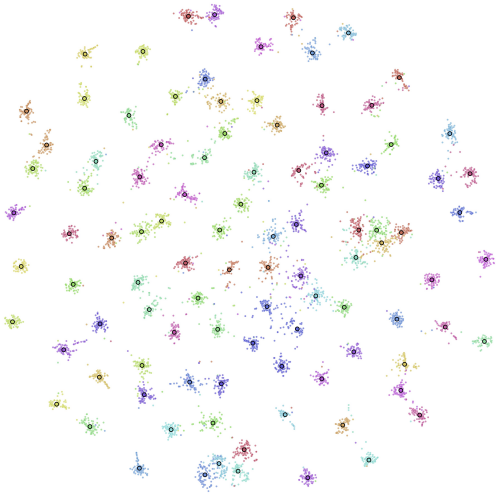
(b) Original test samples with multi-centriod key prototypes



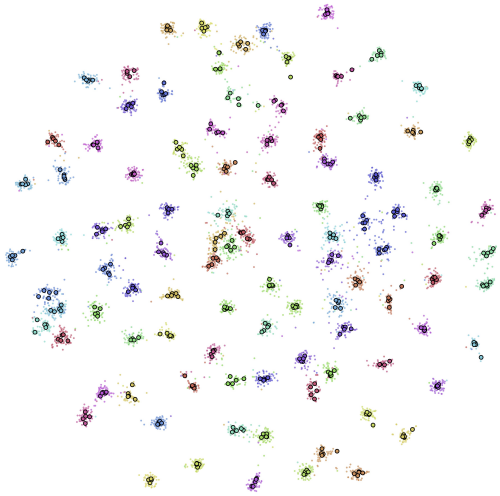
(c) Prompted test samples with key prototypes



(d) Prompted test samples with multi-centroid key prototypes



(e) Prompted test samples with multi-centroid value prototypes



(f) Prompted test samples with value prototypes

Figure 8: Visualization for test data in CIFAR-100.