Anytime Continual Learning for Open Vocabulary Classification

Zhen Zhu, Yiming Gong, and Derek Hoiem

University of Illinois at Urbana-Champaign
{zhenzhu4,yimingg8,dhoiem}@illinois.edu

Abstract. We propose an approach for anytime continual learning (AnytimeCL) for open vocabulary image classification. The AnytimeCL problem aims to break away from batch training and rigid models by requiring that a system can predict any set of labels at any time and efficiently update and improve when receiving one or more training samples at any time. Despite the challenging goal, we achieve substantial improvements over recent methods. We propose a dynamic weighting between predictions of a partially fine-tuned model and a fixed open vocabulary model that enables continual improvement when training samples are available for a subset of a task's labels. We also propose an attention-weighted PCA compression of training features that reduces storage and computation with little impact to model accuracy. Our methods are validated with experiments that test flexibility of learning and inference.

Keywords: Continual learning · Anytime learning · Open-vocabulary classification

1 Introduction

Continual learning aims to improve a system's capability as it incrementally receives new data and labels, which gains increasing importance with the expanding scale and applications of visual learning. Continual learning for classification is traditionally approached with discrete label spaces. Adding labels or tasks over time changes the problem landscape, thereby increasing the difficulty of learning. The open vocabulary setting, however, frames classification as comparing continuous feature and label embeddings. Any images and textual labels can be embedded, so learning in this setting involves only improving on the existing problem definition, which may be more amenable to continuous improvement. Although open vocabulary models can predict over arbitrary label sets, even models like CLIP [38] that are trained on internet-scale data have unsatisfactory performance on many tasks [48, 49].

Our work aims to continually improve open vocabulary image classifiers as new labeled data is received. We call this "**anytime continual learning**" because the goal is to improve efficiently *any time* new examples are received and to maintain the ability to predict over arbitrary label sets at *any time*.

Recent work by Zhu et al. [64] extends CLIP by training linear classifiers and combining their predictions with those of label vectors from the original text embedding. To enable efficient and distributed learning, linear classifiers are trained for each partition of the feature space defined by online hierarchical clustering. This approach requires storing all training examples, but a new example is efficiently incorporated by training only on the data from the example's nearest cluster. In experiments on their own benchmark designed for open vocabulary continual learning and existing benchmarks, Zhu et al. [64] outperform other recent approaches.

Our approach is to train online, fine-tuning the last transformer block while keeping the label embedding fixed. When a new training sample is received, we fill a class-balanced batch with stored samples and update the last transformer block in a single step. Our experiments show this performs better than retraining a classifier layer, and each new sample can be incorporated in milliseconds. Additionally, we introduce a modified loss, enabling the prediction of "none of the above" when the true label is absent from the candidate set, which improves overall performance.

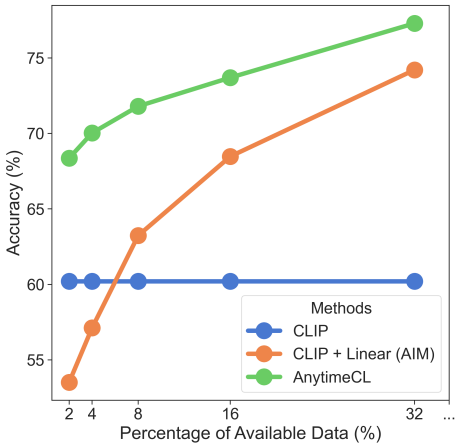

Fig. 1: Our AnytimeCL algorithm can be efficiently updated with each new example and continuously improve. By dynamically weighting predictions between a tunable model and a frozen open vocabulary model, our method can predict over any label set while gaining expertise. This figure shows our method outperforms previous SotA Zhu et al. [64] in every stage in the data incremental setting. Our method also outperforms in other settings like task-incremental and class-incremental.

Whenever a new training example arrives, our method updates an estimate of the tuned and original model's accuracy for the given label. The tuned and original model's predictions are then weighted in proportion to their expected accuracy for each label. This weighting accounts for the growing effectiveness of the tuned model, and greatly outperforms Zhu et al. [64]'s AIM weighting based on whether an example's label is likely to have been seen in training.

The partial fine-tuning approach requires storing either full images that need to be re-encoded during training, or data-consumptive patch-level features at intermediate layers. We apply a per-image weighted PCA, which provides a 30x reduction of data with little impact on prediction accuracy. Such compression could be especially beneficial in a large-scale federated learning setting.

In experiments on the open vocabulary continual learning benchmark proposed by Zhu et al. [64], our approach outperforms under all settings and stages, including data-incremental, class-incremental, and task-incremental learning, and zero shot prediction. Our ablations evaluate the effects of all key design parameters, including the partial fine-tuning, model prediction weighting, batch sampling, and the regularization loss term. We also show that our method can be applied to combine non-open vocabulary models like DINOv2 with CLIP to improve further while maintaining open vocabulary ability.

In summary, our **main contribution** is an open vocabulary continual learning method that can quickly incorporate new training examples received in any order and continually improve while maintaining open vocabulary performance. Our experiments demonstrate that our system's accuracy and efficiency stem from multiple proposed innovations:

- Partial fine-tuning of features with a fixed label embedding;
- Online training with each batch composed of the new training sample and class-balanced stored samples;
- Online learning of per-label accuracy for effective combination of original and tuned model predictions;
- Loss modification to enable "none of the above" prediction, which also stabilizes open vocabulary training;
- Intermediate layer feature compression that reduces storage of training samples and improves speed without much loss to accuracy.

2 Related work

The goal of continual learning (CL) is to continually improve the performance when seeing more data, while online continual learning [1, 3, 8, 25, 35, 37] aims to achieve a good trade-off between performance and learning efficiency. Our problem setup and goals are distinguished by these important characteristics:

- *Flexibly learn*, receiving labeled examples in data-incremental (random order), class incremental (grouped by category), or task incremental (grouped by dataset);
- *Efficiently update* from a single example or a batch of examples;
- *Retain training data*, though potentially in compact, privacy-preserving forms;
- *Cumulatively improve* from new data without forgetting other classes or tasks;
- *Flexibly infer*, predicting over label sets defined at inference time, without rigid task boundaries.

Continual learning approaches can be broadly categorized into regularization [20,24,60], replay/rehearsal methods [25,39,43,57], and parameter isolation or expansion [2, 28, 36, 37, 41, 42, 58, 61]. Regularization techniques generally impose constraints on the learning process to alleviate forgetting. Rehearsal methods involve storing and replaying past data samples during training [5,36,37,43].

So far, when training data can be stored, simple replay methods tend to outperform others, whether limiting the number of stored examples [37] or training computation [36]. Parameter isolation methods maintain learning stability by fixing subsets of parameters [28, 42] or extending model with new parameters [41, 58, 61]. Recently, several works [17–19, 46, 52–54] adopt prompt tuning for continual learning, which can also fall into the parameter expansion category. These methods can be used for stage-wise incremental continual learning, but none demonstrates the efficacy on online incremental learning that is our focus. Prompt tuning provides an alternative to weight-based fine-tuning with less forgetting. We consider weight-tuning approaches here, but preliminary experiments indicate the same strategy is applicable to prompt tuning, with similar accuracy but slower training.

We now focus on those most relevant and influential to our work; see Wang et al. [51] for a comprehensive survey of continual learning.

2.1 CL for open-vocabulary classification

WiSE-FT [55] fine-tunes CLIP encoders on target tasks and averages fine-tuned weights with original weights for robustness to distribution shifts. PAINT [16] shares a similar idea as WiSE-FT but the weight mixing coefficient is found via a held-out set. With such approaches, generality degrades with each increment of continual learning. CLS-ER [4] exponentially averages model weights in different paces for its plastic and stable model to balance learning and forgetting. Closer to our work, ZSCL [63] fine-tunes CLIP encoders using a weight ensemble idea, similar to WiSE-FT, and applies a distillation loss with a large unsupervised dataset to reduce forgetting. Unlike these, we partially fine-tune CLIP with fixed label embeddings and use a dynamically weighted combination of the predictions from the tuned and original models, which enables improvement on target tasks without sacrificing the generality of the original model.

Another class of methods [14,45] use attribute-based recognition [13,23], continually learning attributes and attribute-class relations as a way to generalize from seen to unseen categories. We use CLIP, which provides zero-shot ability by training to match image-text pairs on a large training corpus, instead of the attribute-based approach. In reported results [14, 38, 45] and our own tests, CLIP has much higher zero-shot accuracy (e.g. for the SUN and aPY datasets) than the continual attribute-based methods achieve even after training, supporting the idea of building on vision-language models for open vocabulary image classification.

2.2 CL with constrained computation

Prabhu et al. [36] evaluate a wide variety of continual learning techniques under a setting of retaining data but limiting compute. Compared to all other techniques, they find that simple replay strategies, such as fine-tuning with uniformly random or class-balanced batches, are most effective in this setting. They also provide some evidence that partial-fine tuning of a well-initialized network may

be an efficient and effective approach. In this direction, we find that fine-tuning only the last transformer block of CLIP by performing one mini-batch update per incoming sample works surprisingly well.

2.3 Biologically inspired CL

The complimentary learning systems theory [32] (CLS theory) posits that humans achieve continual learning through the interplay of sparse retrieval-based and dense consolidated memory systems. This has inspired many works in continual learning, e.g. [4,34,64]. The most closely related is Zhu et al. [64], as detailed in the introduction. Many works are also inspired by the idea of wake/sleep phases of learning, where faster updates are required during the wake stage. Recent studies of mammalian memory and learning indicate that consolidation occurs in similar ways via replay during wakefulness and sleep [7, 44, 50]. Our online learning method mixes use of individual examples and consolidated networks for continuous improvement for continuous improvement, reflecting the idea of wakeful consolidation through replay.

2.4 Memory compression

Other work has compressed training data [9], or distilled training data into a network [59], or learned prompts [17–19,46,52–54] instead of maintaining original examples. AQM [9] uses an online VQ-VAE [30] to compress training samples to reduce storage for continual learning. We instead compress intermediate features to improve both storage and computation of partial fine-tuning.

3 Method

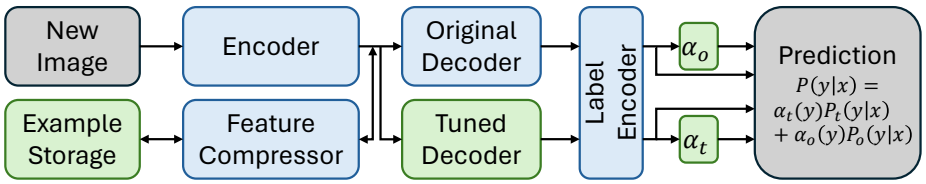

Fig. 2: Overview. On receiving a new training sample, the batch is completed with stored samples, the prediction is made using a label embedding, the tuned decoder and confidence weights (α_o, α_t) are updated in one step, and the new sample is stored. To save space and time, stored examples are encoded and compressed. In testing, the probability of each candidate label is determined by predictions from both decoders and their class-wise confidence weights. Our method enables constant-time updates from new samples while continually improving and maintaining open vocabulary performance. Green blocks are updated during training; blue blocks are not updated.

Our system receives training examples $(x, y, \mathcal{Y})$ one by one to update its models, in order to continually improve in prediction of y given $(x, \mathcal{Y})$, where x is an input image, y is its target label, and $\mathcal{Y}$ is its set of candidate labels.

Fig. 2 offers a system overview aimed at enhancing open vocabulary image classifiers by integrating a tuned model for learning target tasks with a frozen original model. The tuned model uses the same encoder as the original but incorporates a trainable decoder. For an image x, both the tuned model and the original model produce the probabilities for all candidate labels, denoted as $P_t(y|x)$ and $P_o(y|x)$. The final probability for the image is weighted through our Online Class-wise Weighting (OCW, Sec. 3.2):

$$P(y|x) = \alpha_o(y)P_t(y|x) + \alpha_t(y)P_o(y|x), \tag{1}$$

where $\alpha_o(y)$ and $\alpha_t(y)$ are the relative expected accuracies for the original and tuned models for label y. During training, new samples are encoded to intermediate features (feature vectors for patches plus a CLS token), optionally compressed (Sec. 3.3), and stored, for reuse in future steps.

3.1 Models

Original model. In our experiments, the original model is the publicly available CLIP [38] ViT model that has been trained contrastively on image-text pairs. In our experiments, the shared encoder is all but the last transformer block, and the original decoder is the last block. The CLIP model produces a probability for label y for image x given a set of candidate text labels $\mathcal{Y}$ based on the dot product of image embedding e_x (CLS token) and text embedding e_y:

$$P_o(y|x) = \frac{\exp(100 \cdot \cos(e_x, e_y))}{\sum_{y_k \in \mathcal{Y}} \exp(100 \cdot \cos(e_x, e_{y_k}))}. \tag{2}$$

Tuned model. With each new sample, our goal is to efficiently update the tuned model to improve accuracy in received labels while maintaining general features that are effective for future learning. We tune only the last image transformer block while keeping the label embedding fixed, which helps the features to stay correlated with the text modality and reduces overfitting to received labels. Note that more blocks can be tuned, which could enable better performance but with increased computation. The tuned decoder is initialized with the weights of the original decoder.

Given a new sample, we form a batch of that sample and a class-balanced sampling of stored training samples. Specifically, we first determine the class count for selection: for a batch size of B and all seen labels $\mathcal{Y}_t$, we uniformly select $\min(B-1, |\mathcal{Y}_t|)$ classes from $\mathcal{Y}_t$, where $|\cdot|$ indicates the length of a set. Then we uniformly sample an equal number of instances from each chosen class to form the batch. This class-balanced sampling ensures continual retention and improvement of prediction of training labels. Then, the tuned model is updated with one iteration based on a cross-entropy loss.

Additionally, we develop a regularization loss that helps with performance. The idea is that, if the true label is not in the label candidates, a low score should be predicted for every candidate label. We implement this with an "other" option within the candidate set, but since "other" does not have an appearance, we model it with just a learnable bias term. The combined loss for training the tuned model is therefore:

$$\mathcal{L}(x, y, \mathcal{Y}) = \mathcal{L}_{\text{ce}}(x, y, \mathcal{Y} \cup \text{other}) + \beta \mathcal{L}_{\text{ce}}(x, \text{other}, (\mathcal{Y} \cup \text{other}) \setminus y), \tag{3}$$

where β is a hyperparameter balancing the two loss terms, set to 0.1 based on validation experiments.

3.2 Online Class-wise Weighting

We estimate the probability of each candidate label by weighted voting of the tuned and original models (Eq. 1). We would like to assign more weight to the model that is more likely to be correct for a given label. There are two big problems: 1) training samples would provide a highly biased estimate of correctness for the tuned model; 2) maintaining a held out set or performing cross-validation would either reduce available training samples or be impractically slow. Our solution is to use each training sample, before update, to update our estimate of the likelihood of correctness for its label based on the tuned and original predictions.

Let $c_t(y)$ and $c_o(y)$ be the estimated accuracy of the tuned and original model for label y. We apply an exponential moving average (EMA) updating method to estimate them online, ensuring the estimations are reliable when evaluated at any time, concurring with our anytime continual learning goals. Assuming the EMA decay is set to η (=0.99 in our experiments), the estimated accuracy of the tuned model at the current step is:

$$c_t(y) = \eta \hat{c}_t(y) + (1 - \eta) \mathbb{1}[y_t(x) = y]. \tag{4}$$

Here, $\hat{c}_t(y)$ is the estimated accuracy of label y in the previous step; $y_t(x)$ denotes the predicted label of the tuned model for x. Since the exponential moving average depends on past values, we compute $c_t(y)$ as the average accuracy for the first $\lfloor \frac{1}{1-\eta} \rfloor$ samples. $c_o(y)$ is updated in the same way.

After getting $c_t(y)$ and $c_o(y)$, the weights of the two models are:

$$\alpha_t(y) = \frac{c_t(y)}{c_t(y) + c_o(y) + \epsilon}, \qquad \alpha_o(y) = 1 - \alpha_t(y). \tag{5}$$

Here, ϵ is a very small number (1e-8) to prevent division by zero. For labels not seen by the tuned model, we set $\alpha_t(y) = 0$, so $\alpha_o(y) = 1$.

3.3 Storage efficiency and privacy

Partial tuning of our model requires either storing each image or storing the features (or tokens) that feed into the tuned portion of the model. Storing images

has disadvantages of lack of privacy and inefficiency in space and computation, due to need to re-encode in training. Storing features alleviates some of these problems, but still uses much memory or storage. Therefore, we investigate how to compress the training features, aiming to increase storage and computational efficiency while maintaining training effectiveness. The compressed features also provide more privacy than storing original images, though we do not investigate how well the images can be recovered from compressed features.

Well-trained networks learn data-efficient representations that are difficult to compress. Indeed, if we try to compress feature vectors with VQ-VAE [30] or PCA (principal component analysis) trained on a dataset, we are not able to achieve any meaningful compression without great loss in training performance. The features within *each image*, however, contain many redundancies. We, therefore, compute PCA vectors on the features in each image and store those vectors along with the coefficients of each feature vector. Further, not all tokens are equally important for prediction. Hence, we train a per-image attention-weighted PCA, weighted by attention between each token and the CLS token. Finally, we can compress further by storing min/max floating point values for each vector and for the coefficients and quantizing them to 8-bit or 16-bit unsigned integers. See the supplemental material for details. By storing only five PCA vectors and their coefficients this way, we can reduce the storage of fifty 768-dim tokens (7×7 patch tokens + CLS token) from 153K bytes to 5K bytes with less than 1% difference in prediction accuracy.

4 Experiments

Using the setup from [64], we sequentially receive training samples for a target task in a **data-incremental** (random ordering) or **class-incremental** (class-sorted ordering). Or, in a **task-incremental** setting, we receive all examples for each task sequentially. Within each set, we incorporate new examples **one by one** in an online fashion. After receiving each set, the model is evaluated on the entire task (including classes not yet seen in training) for data- or class-incremental, or on all tasks (including tasks not yet seen in training) for task-incremental.

Our main experiments use the same setup as Zhu et al. [64], which are designed to test flexible learning and flexible inference. A **target task** contains a subset of seen labels while a **novel task** contains none of these labels. Target tasks are CIFAR100 [22], SUN397 [56], FGVCAircraft [27], EuroSAT [15], OxfordIIITPets [33], StanfordCars [21], Food101 [6], and Flowers102 [29]. Novel tasks are ImageNet [40], UCF101 [47], and DTD [10]. Training examples are received for target tasks, but not for novel tasks. Under this setup, we have 226,080 training samples and 1,034 classes in total.

- **Task Incremental Learning** updates one model with each of the eight target tasks sequentially.
- **Class Incremental Learning** updates a per-task model with examples from a subset (one-fifth) of classes for a stage.

- **Data Incremental Learning** updates a per-task model with a random subset of the data in eight stages: 2%, 4%, 8%, 16%, 32%, 64%, 100%.

After each stage, accuracy is averaged across all tasks (including tasks/classes not yet seen).

Additionally, we provide a **union data incremental** scenario where we mix all target tasks together and union all target labels, mainly for hyperparameter selection and ablation experiments. **Flexible inference** is evaluated after the task-incremental learning for prediction over novel tasks and sets of candidate labels $\mathcal{Y}$: zero-shot, union of all target tasks/labels and zero-shot tasks/labels, and a mix of some target and zero-shot tasks/labels. See [64] for details. The union and mixed settings are most challenging because they require effective combined use of both the original and tuned models, without any task identifiers or indication whether an example's label is in the tuned model's domain.

Implementation details. We employ the ViT-B/32 model from CLIP as our network backbone. We use AdamW [26] as the optimizer with a weight decay of 0.05, and we adhere to CLIP's standard preprocessing without additional data augmentations. For offline training, following [11], we set the learning rate at 6e-4 and the batch size at 2048, employing a cosine annealing scheduler for the learning rate with a minimum rate of 1e-6. For online training, we use a batch size of 32 and a learning rate of $9.375e = \frac{32 \times 6e-4}{2048}$, based on adjustments from the offline training hyperparameters. More method and training details are in the supplemental.

4.1 Main result

In this evaluation, we compare our method with the previous SotA Zhu et al. [64] (CLIP+LinProbe (AIM-Emb)) under the task-, class-, data-incremental scenarios, as well as flexible inference. From all plots in Fig. 3, our AnytimeCL method consistently outperforms CLIP+LinProbe (AIM). For task incremental, the improvement is mainly due to the partial finetuning with fixed label embeddings. To provide a more direct comparison with Zhu et al. [64], we train our approach only after receiving all data of a task (*e.g.*, 100% CIFAR100) and term this as AnytimeCL (Offline), which also shows steady improvements over Zhu et al. [64]. For early stages in class incremental and data incremental, the performance improvement mainly comes from the advantage of our online class-wise weighting method over AIM, which only considers whether the testing sample is likely to be from a known class.

4.2 Comparison under MTIL task incremental learning [63]

We compare the performance of AnytimeCL with Zhu et al. [64] and ZSCL [63] under the task incremental learning setting as introduced in ZSCL. Results are summarized in Tab. 1, using the Transfer, Last, and Average metrics from [63] for comparison. Improvements over CLIP are denoted in green under the Δ columns, while declines are marked in red. Our method consistently achieves

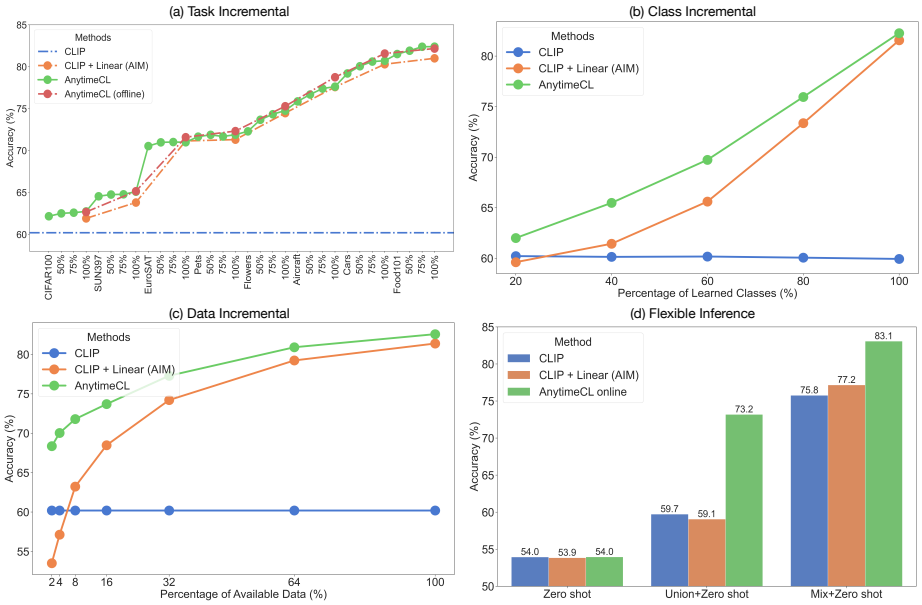

Fig. 3: Performance comparison of CLIP, CLIP + Linear (AIM), and AnytimeCL variants in task incremental (a); class incremental (b); data incremental (c); and flexible inference (d) settings. In (a), the online approach is represented by a solid line, while offline methods are depicted with dashed lines, assessed when the online algorithms receive 25%, 50%, 75%, and 100% data of a task, labeling tasks at the 25% point.

zero loss in Transfer in both experiments, indicating an absence of forgetting. On this benchmark, our method performs comparably to LinProbe (AIM) and outperforms other methods. This experimental setup does not fully capture our method's key advantages of online training, ability for data and class incremental learning, and open vocabulary prediction. Also, in this setting, a simple weighting method is effective — to use the tuned model's prediction for tasks with all candidate labels in the training set, or the original model's prediction otherwise.

4.3 Training features compression

We test our per-image attention-weighted PCA compresssion for training features (Table 2). We compare the data size, average full time to process one 32-sample batch (including data loading, uncompressing, and forward/backward training passes), and final accuracy after completing fine-tuning. This test is on CIFAR100. Compared to processing the full image or full features, using our compressed features saves 30x the storage while achieving nearly the same accuracy. We were not able to achieve good accuracy when compressing with VQ-VAE, even when using large vocabularies, nor when using PCA vectors computed over the entire dataset. Computing PCA vectors per instance (over the 50 tokens) and weighting the tokens by their CLS-attention each improve the tuned accu-

Method	Transfer	Δ	Avg.	Δ	Last	Δ
CLIP	69.4	0.0	65.3	0.0	65.3	0.0
LwF [24]	56.9	-12.5	64.7	-0.6	74.6	+9.3
iCaRL [39]	50.4	-19.0	65.7	+0.4	80.1	+14.8
WiSE-FT [55]	52.3	-17.1	60.7	-4.6	77.7	+12.4
ZSCL [63]	68.1	-1.3	75.4	+10.1	83.6	+18.3
TreeProbe (AIM) (50k) [64]	69.3	-0.1	75.9	+10.6	85.5	+20.2
LinProbe (AIM) [64]	69.3	-0.1	**77.1**	**+11.8**	86.0	+20.7
AnytimeCL	**69.4**	**0**	77.0	+11.7	85.8	+20.5
AnytimeCL (Offline)	**69.4**	**0**	77.0	+11.7	**86.2**	+20.9

Table 1: Comparison of different methods on MTIL in Order I from ZSCL [63]. CLIP is our tested zero-shot performance. All other results are taken Zhu et al. [64] and ZSCL [63]'s papers. "Transfer" evaluates the model's performance on zero-shot tasks; "Last" is the averaged accuracy on all target tasks after finishing the final task while "Avg." computes the average task performance on all training stages.

racy. Quantizing the PCA vectors and coefficients further saves space at a small loss to accuracy. Using the compressed features also gives a nearly 2x speedup vs. using the full features (because the compressed features fit entirely in memory) and 6x vs. using the full image, though all times are fast. We provide more method details in the supplemental along with a comparison of using different per-instance PCA compression methods under the union data incremental scenario, to show how these compression methods influence the result at various timesteps.

Compression	KB/example	ms/batch	FT Accuracy
Full image	150.5	43.9	77.8
Full features	153.6	25.6	77.8
VQ	0.2	4.5	64.8
Dataset-wide PCA (200 components)	40.2	7.6	76.4
Per-instance PCA (5 components)	19.4	7.7	74.6
+ CLS-weight (5 components)	19.4	8.9	77.9
+ int-quantization (5 components)	5.3	13.9	77.5

Table 2: Comparison of different compression methods in terms of memory usage, processing speed, and test accuracy on CIFAR100. See supplementary material for details on how to get ms/batch and FT Accuracy. We use the same seed for all methods.

4.4 Ablations

In the following, we ablate several important factors that contribute to our method. Additional ablations and results are provided in the supplemental.

Weighting Strategies. Weighting is vital for our dual decoder approach. We compare several ways to compute the weights: 1) CLIP: namely $\alpha_t(y) = 0$ for

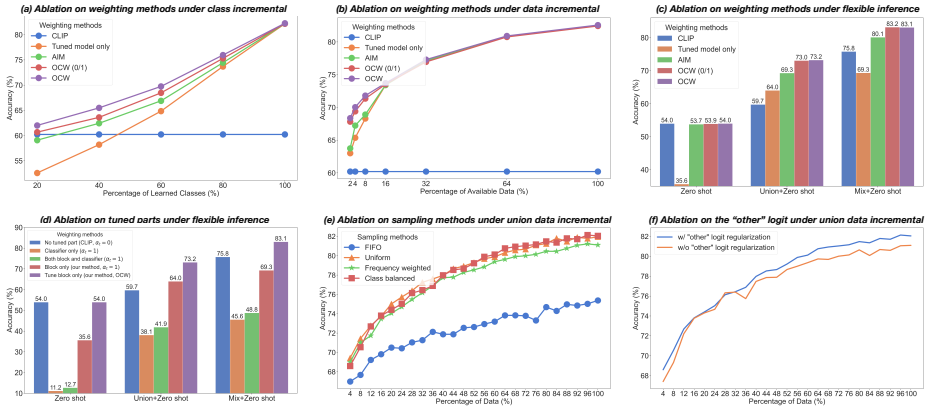

Fig. 4: Ablation results. Best viewed in color and zoomed-in.

any images; 2) Tuned model only: $\alpha_t(y) = 1$ for any images; 3) AIM [64]; 4) OCW (0/1): a variant of OCW where we round $\alpha_t(y)$ to 0 or 1 to use either the original or tuned model; 5) Our proposed OCW. We partially finetune the decoder with fixed label embeddings and combine the tuned model with the original model using different weighting strategies. The results are shown in Fig. 4 (a), (b) and (c), under data-, class-incremental and flexible inference. OCW performs the best of all methods in every setting and stage. Notably, OCW enables improvements even in the early stages when limited data is available for a class or task and better handles tasks that mix novel and target labels.

Tuned parts. Our proposed method tunes the last transformer block while keeping the label encoder fixed. In Fig. 4 (d), we compare this approach to alternatives of tuning only the label encoder, both the block and the label encoder, or neither of them, under the flexible inference test. When only using the tuned model for comparison ($\alpha_t = 1$), fine tuning only the last transformer best retains predictive ability for novel labels. Further analysis under the task incremental scenario is provided in the supplemental material.

Sampling methods. We compare different methods for sampling from the stored samples. FIFO cycles through samples in order of first appearance, "uniform" randomly draws samples for each batch, class-balanced (which we use in all other experiments) samples classes, and frequency weighted sampling (FWS) samples based on how many times a given sample has been batched in training. Class-balanced and uniform are similar in practice, and perform best (Fig. 4 (e)). This result is consistent with Prabhu et al. [36]'s finding that simple sampling methods outperform complex ones under a computational budget. See supplemental for more details.

The "other" logit regularization. The ablation experiments depicted in Fig. 4 (f) assess the impact of "other" logit regularization in the union data incremental scenario. The results demonstrate consistent enhancements when this regularization is applied, compared to its absence.

4.5 Scalability

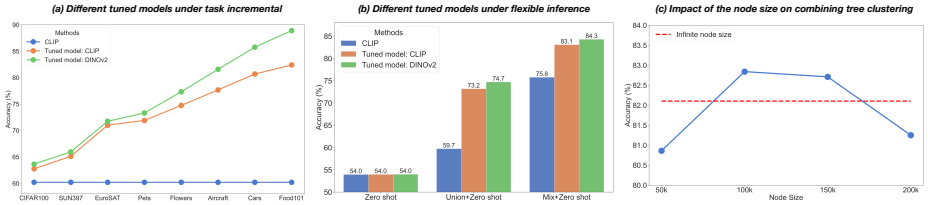

Fig. 5: (a) & (b): The "CLIP" method refers to the original model. For Tuned model: DINOv2, we use the ViT-B/14 checkpoint; (c): Infinite node size indicates only one tuned model regardless of the number of samples received.

DINOv2 as the Tuned Model. Using self-supervised DINOv2 [31] as our tuned model showcases our method's adaptability beyond the original CLIP model [64]. Our approach allows for tuning the feature embedding space while keeping label embeddings fixed, enabling the incorporation of diverse encoders through an additional linear layer for dimension matching. This layer is trained in joint with the tuned decoder. As illustrated in Fig. 5 (a), replacing the tuned model with DINOv2 results in consistent performance improvements at every stage, with a notably steeper improvement curve in later stages. Furthermore, Fig. 5 (b) demonstrates that this modification preserves Zero-shot performance while significantly boosting Union+Zero-shot and Mix+Zero-shot outcomes, attributed to DINOv2's enhanced performance on target tasks.

Tree clustering models. Zhu et al. [64] propose tree-based clustering as a way to limit the training time needed to incorporate new examples. Our method can already incorporate an example much faster in a way that does not depend on the number of training samples, but the clustering idea is still intriguing as a path to scalability in storage, memory, and model capacity. In Fig. 5 (b), we evaluate different node capacities and find that moderate-sized partitions may slightly increase performance. In these experiments, each node stores a copy of the last CLIP transformer block and updates it based on data that it receives.

5 Discussion

Our AnytimeCL approach shows promise of breaking free of staged train and deploy processes, with many benefits to applications that constantly or sporadically receive new data and have customizable or evolving label spaces.

The *anytime* has three main parts: (1) being able to do a task even when training data do not fully cover the label space for that task; (2) being able to incorporate a new training example quickly; and (3) not losing the benefit of older training data. Our key to the first part is using complimentary models. For that we are motivated by CLS theory [32] and more specifically adopt the

framework of Zhu et al. [64], improving it with our prediction weighting method and by partially fine-tuning the encoder. Our key to the second part is that we do not need to incorporate one million examples quickly, just one. This reframes the computationally budgeted CL problem of Prabhu et al. [36], who aim to incorporate a new large batch of examples in time that allows visiting only a portion of the existing data. For some settings, our goal is more practical, as it enables an opportunistic approach that interleaves pure online learning during busy times and offline learning when time allows. While humans are often held up as excellent continual learners, they also learn one thing fast but many things slowly over time and make use of restful wake and sleep to continue learning. The effectiveness of our online learning is also due, in part, to leveraging a strong vision-language model, just recently possible. The key to the third goal is simple — do not throw out the data. If one cares about privacy and storage, then finding ways of compressing and preserving privacy, like our PCA-based approach, may be better than trying to retain the benefits of a large dataset without one.

Potential directions for future work:

- Beyond classification: Our partial fine-tuning approach is applicable to other tasks, such as semantic segmentation, visual question answering, and object detection. Our weighting method is applicable to open vocabulary detection and segmentation.
- Multi-model inference: Extending our approach to multiple models, including task-specific models, would further extend flexible learning and model re-use opportunities.
- Scalability: We expect that tree-based data clustering and training one model per cluster, as in [64], provides a good mechanism for scalability. Larger scale experiments, involving dozens of datasets and many millions of examples, are needed to fully test this.
- Federated learning: Combining tree-based clustering and feature compression, we can encode training examples in the client or central server and transmit to a node that stores and updates the tunable block and its data. This enables fully distributed training on inexpensive nodes.

6 Conclusion

We propose an effective approach for anytime continual learning of open vocabulary image classification. The main innovation is dynamic class-senstive weighting for combining predictions of open-vocabulary and example-based tuned models, and we also find that partial fine-tuning with sufficiently small learning rates enables a surprisingly effective online learner. We offer a per-image attention-weighted PCA approach to compress features, with benefits to storage, computation, and privacy. Our experiments show that our approach is very promising, and further exploration is merited to more fully understand its effective scope and limitations.

Acknowledgement

This work is supported in part by ONR award N00014-21-1-2705, ONR award N00014-23-1-2383, and U.S. DARPA ECOLE Program No. #HR00112390060. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, ONR, or the U.S. Government.

References

1. Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Lin, M., Charlin, L., Tuytelaars, T.: Online continual learning with maximally interfered retrieval. CoRR (2019)
2. Aljundi, R., Chakravarty, P., Tuytelaars, T.: Expert gate: Lifelong learning with a network of experts. In: CVPR (2017)
3. Aljundi, R., Lin, M., Goujaud, B., Bengio, Y.: Gradient based sample selection for online continual learning. In: NeurIPS (2019)
4. Arani, E., Sarfraz, F., Zonooz, B.: Learning fast, learning slow: A general continual learning method based on complementary learning system. In: ICLR (2022)
5. Bang, J., Kim, H., Yoo, Y., Ha, J., Choi, J.: Rainbow memory: Continual learning with a memory of diverse samples. In: CVPR (2021)
6. Bossard, L., Guillaumin, M., Gool, L.V.: Food-101 - mining discriminative components with random forests. In: ECCV (2014)
7. Brodt, S., Inostroza, M., Niethard, N., Born, J.: Sleep—a brain-state serving systems memory consolidation. Neuron (2023)
8. Caccia, L., Aljundi, R., Asadi, N., Tuytelaars, T., Pineau, J., Belilovsky, E.: New insights on reducing abrupt representation change in online continual learning. In: ICLR (2022)
9. Caccia, L., Belilovsky, E., Caccia, M., Pineau, J.: Online learned continual compression with adaptive quantization modules. In: ICML (2020)
10. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , Vedaldi, A.: Describing textures in the wild. In: CVPR (2014)
11. Dong, X., Bao, J., Zhang, T., Chen, D., Gu, S., Zhang, W., Yuan, L., Chen, D., Wen, F., Yu, N.: CLIP itself is a strong fine-tuner: Achieving 85.7% and 88.0% top-1 accuracy with vit-b and vit-l on imagenet. CoRR (2022)
12. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: ICLR (2021)
13. Farhadi, A., Endres, I., Hoiem, D., Forsyth, D.: Describing objects by their attributes. In: CVPR. IEEE (2009)
14. Gautam, C., Parameswaran, S., Mishra, A., Sundaram, S.: Online lifelong generalized zero-shot learning. Neural networks (2021)
15. Helber, P., Bischke, B., Dengel, A., Borth, D.: Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (2019)
16. Ilharco, G., Wortsman, M., Gadre, S.Y., Song, S., Hajishirzi, H., Kornblith, S., Farhadi, A., Schmidt, L.: Patching open-vocabulary models by interpolating weights. In: NeurIPS (2022)

17. Jin, W., Cheng, Y., Shen, Y., Chen, W., Ren, X.: A good prompt is worth millions of parameters: Low-resource prompt-based learning for vision-language models. In: Proc. ACL (2022)
18. Khattak, M.U., Rasheed, H.A., Maaz, M., Khan, S.H., Khan, F.S.: Maple: Multi-modal prompt learning. In: CVPR (2023)
19. Khattak, M.U., Wasim, S.T., Naseer, M., Khan, S., Yang, M., Khan, F.S.: Self-regulating prompts: Foundational model adaptation without forgetting. In: ICCV (2023)
20. Kirkpatrick, J., Pascanu, R., Rabinowitz, N.C., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R.: Overcoming catastrophic forgetting in neural networks. CoRR (2016)
21. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3d object representations for fine-grained categorization. In: 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13) (2013)
22. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto (2009)
23. Lampert, C.H., Nickisch, H., Harmeling, S.: Attribute-based classification for zero-shot visual object categorization. IEEE TPAMI (2014)
24. Li, Z., Hoiem, D.: Learning without forgetting. In: ECCV (2016)
25. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continual learning. In: NeurIPS (2017)
26. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: ICLR (2019)
27. Maji, S., Rahtu, E., Kannala, J., Blaschko, M.B., Vedaldi, A.: Fine-grained visual classification of aircraft. CoRR (2013)
28. Mallya, A., Lazebnik, S.: Packnet: Adding multiple tasks to a single network by iterative pruning. In: CVPR (2018)
29. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing (2008)
30. van den Oord, A., Vinyals, O., kavukcuoglu, k.: Neural discrete representation learning. In: NeurIPS (2017)
31. Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Galuba, W., Howes, R., Huang, P., Li, S., Misra, I., Rabbat, M.G., Sharma, V., Synnaeve, G., Xu, H., Jégou, H., Mairal, J., Labatut, P., Joulin, A., Bojanowski, P.: Dinov2: Learning robust visual features without supervision. CoRR (2023)
32. O'Reilly, R.C., Bhattacharyya, R., Howard, M.D., Ketz, N.: Complementary learning systems. Cogn. Sci. (2014)
33. Parkhi, O.M., Vedaldi, A., Zisserman, A., Jawahar, C.V.: Cats and dogs. In: CVPR (2012)
34. Pham, Q., Liu, C., Hoi, S.: Dualnet: Continual learning, fast and slow. In: NeurIPS (2021)
35. Prabhu, A., Cai, Z., Dokania, P.K., Torr, P.H.S., Koltun, V., Sener, O.: Online continual learning without the storage constraint. CoRR (2023)
36. Prabhu, A., Hammoud, H.A.A.K., Dokania, P.K., Torr, P.H.S., Lim, S., Ghanem, B., Bibi, A.: Computationally budgeted continual learning: What does matter? CVPR (2023)
37. Prabhu, A., Torr, P., Dokania, P.: Gdumb: A simple approach that questions our progress in continual learning. In: ECCV (2020)

38. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. In: ICML (2021)
39. Rebuffi, S., Kolesnikov, A., Sperl, G., Lampert, C.H.: icarl: Incremental classifier and representation learning. In: CVPR (2017)
40. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Fei-Fei, L.: Imagenet large scale visual recognition challenge. IJCV (2015)
41. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv:1606.04671 (2016)
42. Serrà, J., Suris, D., Miron, M., Karatzoglou, A.: Overcoming catastrophic forgetting with hard attention to the task. In: ICML (2018)
43. Shin, H., Lee, J.K., Kim, J., Kim, J.: Continual learning with deep generative replay. In: NeurIPS (2017)
44. Siegel, J.M.: Memory consolidation is similar in waking and sleep. Curr Sleep Med Rep (2021)
45. Skorokhodov, I., Elhoseiny, M.: Class normalization for (continual)? generalized zero-shot learning. In: ICLR (2021)
46. Smith, J.S., Karlinsky, L., Gutta, V., Cascante-Bonilla, P., Kim, D., Arbelle, A., Panda, R., Feris, R., Kira, Z.: Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In: CVPR (2023)
47. Soomro, K., Zamir, A.R., Shah, M.: UCF101: A dataset of 101 human actions classes from videos in the wild. CoRR (2012)
48. Tong, S., Jones, E., Steinhardt, J.: Mass-producing failures of multimodal systems with language models. In: NeurIPS (2023)
49. Tong, S., Liu, Z., Zhai, Y., Ma, Y., LeCun, Y., Xie, S.: Eyes wide shut? exploring the visual shortcomings of multimodal llms. CoRR (2024)
50. Wamsley, E.J.: Offline memory consolidation during waking rest. Nature Reviews Psychology (2022)
51. Wang, L., Zhang, X., Su, H., Zhu, J.: A comprehensive survey of continual learning: Theory, method and application. CoRR (2023)
52. Wang, Y., Huang, Z., Hong, X.: S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning. In: NeurIPS (2022)
53. Wang, Z., Zhang, Z., Lee, C., Zhang, H., Sun, R., Ren, X., Su, G., Perot, V., Dy, J.G., Pfister, T.: Learning to prompt for continual learning. In: CVPR (2022)
54. Wang, Z., Zhang, Z., adn Ruoxi Sun, S.E., Zhang, H., Ren, C.Y.L.X., Su, G., Perot, V., Dy, J., Pfister, T.: Dualprompt: Complementary prompting for rehearsal-free continual learning. In: ECCV (2022)
55. Wortsman, M., Ilharco, G., Kim, J.W., Li, M., Kornblith, S., Roelofs, R., Lopes, R.G., Hajishirzi, H., Farhadi, A., Namkoong, H., Schmidt, L.: Robust fine-tuning of zero-shot models. In: CVPR (2022)
56. Xiao, J., Hays, J., Ehinger, K.A., Oliva, A., Torralba, A.: Sun database: Large-scale scene recognition from abbey to zoo. In: CVPR (2010)
57. Yan, S., Xie, J., He, X.: DER: dynamically expandable representation for class incremental learning. In: CVPR (2021)
58. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. In: ICLR (2018)
59. Yu, R., Liu, S., Wang, X.: Dataset distillation: A comprehensive review (2023)
60. Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. In: ICML. pp. 3987–3995 (2017)

61. Zhang, J.O., Sax, A., Zamir, A., Guibas, L.J., Malik, J.: Side-tuning: A baseline for network adaptation via additive side networks. In: ECCV (2020)
62. Zheng, C., Vedaldi, A.: Online clustered codebook. In: ICCV (2023)
63. Zheng, Z., Ma, M., Wang, K., Qin, Z., Yue, X., You, Y.: Preventing zero-shot transfer degradation in continual learning of vision-language models. CoRR (2023)
64. Zhu, Z., Lyu, W., Xiao, Y., Hoiem, D.: Continual learning in open-vocabulary classification with complementary memory systems. CoRR (2023)

Anytime Continual Learning for Open Vocabulary Classification

Zhen Zhu, Yiming Gong, and Derek Hoiem

University of Illinois at Urbana-Champaign
{zhenzhu4,yimingg8,dhoiem}@illinois.edu

S-1 Summary of contents

The supplemental file contains:

- **Additional method details (Sec. S-2)**, including compression method details; the implementation details of the frequency weighted sampler.
- **Additional ablation results (Sec. S-3)**, regarding the online learning batch size selection; the EMA decay (η) for accuracy estimation in OCW; loss balancing parameter β in Eq. 3; the number of saved components for per-instance PCA compression; different per-instance PCA compression methods in the union data incremental scenario.
- **Additional results to the main draft (Sec. S-4)**, including DINOv2 [31] vs. CLIP [38] tuned model on class, data, and union data incremental; weighting method ablation on task incremental; tuned parts ablation on task incremental; detailed results of our method on the ZSCL [63] MTIL task incremental benchmark.

S-2 Additional method details

S-2.1 Compression method detail

In this section, we complement more details of methods and implementation of Tab. 2 in the main paper.

Full Image. CLIP reshapes the input image to 224×224 and then creates 32×32 non-overlapping image patches, resulting in 7×7 patch tokens $\mathbf{P} \in \mathbb{R}^{49 \times D}$ for the transformer, where D (=768 in our case) represents the feature vector dimension. Additionally, a CLS token $\mathbf{C} \in \mathbb{R}^{1 \times D}$ is attached to the start of the token sequence. The CLS token after being processed through the whole transformer is normally followed by a classification layer in ViT [12] or directly used for calculating cosine similarity with label embeddings in CLIP [38] (See Eq. 2). For fair comparison with other methods in Tab. 2, we partially fine-tune the last layer of the transformer block without tuning the label embeddings. Processing a batch includes image loading, processing, a complete forward pass, and a backward pass restricted to the final transformer block.

Full Features. We pre-compute the intermediate features $\mathbf{f} = [\mathbf{C}; \mathbf{P}] \in \mathbb{R}^{50 \times D}$ before the final layer on all data and then store them to disk. For fine-tuning, we

load stored features back from disk to RAM and feed them directly into the final transformer block, which is the only block tuned. Processing a batch includes loading intermediate features, forward and backward pass of the final block.

VQ. For Vector Quantization, we pre-train a codebook on intermediate features before the final layer using MSE loss between the quantized features and the original features. We adopt Zheng et al. [62]'s method to enhance VQ performance by promoting the use of more codebook vectors. Then, we process all images in the dataset to obtain the corresponding codebook indices. We store the codebook and indices of all samples to disk. Similar to Full Features, when fine-tuning, data processing involves loading a codebook and integer indices of intermediate features into RAM for feature reconstruction. To process a single batch, we need loading the codebook and indices, feature reconstruction by retrieving codewords from the codebook according to the indices, forward and backward pass of the final layer.

PCA. To compress $\mathbf{f}$, we first center $\mathbf{f}$ by subtracting the mean values of each token: $\hat{\mathbf{f}} = \mathbf{f} - \mu$, where μ represent the mean values of all tokens. We follow scikit-learn to perform SVD: $\hat{\mathbf{f}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. The reduced form of $\hat{\mathbf{f}}$ can be approximated by $\hat{\mathbf{f}} \approx \mathbf{U}_n\mathbf{\Sigma}_n\mathbf{V}_n^T$, where n is the number of principal components/singular values chosen. $\mathbf{U}_n\mathbf{\Sigma}_n$ is the PCA coefficient matrix and $\mathbf{V}_n^T$ is the PCA component matrix. Ideally, we want to find a small n so that the accuracy after fine-tuning is reasonably good. We store $\mathbf{U}_n\mathbf{\Sigma}_n$, $\mathbf{V}_n$ and μ to the disk for reconstructing $\mathbf{f}$ during training: $\mathbf{U}_n\mathbf{\Sigma}_n\mathbf{V}_n^T + \mu$.

Dataset-wide PCA. Since $\mathbf{f}$ is of shape $50 \times D$ and in our case $D = 768 > 50$, it makes a higher compression rate to compress along the vector dimension. For dataset-wide PCA, we concatenate the intermediate features from all samples together along the dimension of the 50 tokens and then apply PCA on the whole concatenated feature. Due to hardware limitation of RAM capacity, processing all data in a single pass is not feasible. Therefore, we divide all data into chunks of 5,000 samples each. PCA is performed on each chunk and we store feature means, PCA coefficients, and components to disk. Processing a single batch involves loading these data from disk to RAM, reconstructing features via PCA, and processing them through the final layer (both forward and backward pass).

Per-instance PCA. For this method, PCA is applied on the 50 tokens of *each sample*. We store the feature means, PCA coefficients, and components to disk. Procedures required to process a batch are similar as dataset-wide PCA.

CLS-weight. We use the CLS token $\mathbf{C}$ for classification purposes. The idea is that some of the patch tokens $\mathbf{P}$ are quite similar to this CLS token and could be very important for the classification task. To take advantage of this, we adjust the importance (or weight) of these patch tokens based on how similar they are to the CLS token. By doing so, we can potentially compress the information more effectively without losing important details necessary for accurate classification. Specifically, these similarities are obtained through:

$$S = \text{Softmax}(\text{Matmul}(\text{LayerNorm}(\mathbf{C}), \text{LayerNorm}(\mathbf{P}^T))). \tag{S1}$$

Here, we use the layer normalization from the last transformer block. Then, the patch tokens are re-weighted by: $S \cdot \mathbf{P}$. Following this reweighting, per-instance PCA is applied to the adjusted intermediate features: $f_{\text{weighted}} = [\mathbf{C}; S \cdot \mathbf{P}]$.

Int-quantization. To further reduce memory usage, we convert principal components from 32-bit floats to 8-bit integers, potentially reducing memory by approximately fourfold. Suppose a principal component is V, we map it to uint8 (0–255) after performing min-max normalization:

$$\hat{V} = \text{Int}(255 * (\frac{V - V_{\min}}{V_{\max} - V_{\min}})), \tag{S2}$$

where $V_{\min}$ and $V_{\max}$ are the minimum and maximum values of vector V; $\text{Int}(\cdot)$ means converting the values of a vector to integers. After the quantization process, we store the min, max values, and the uint8 vector. Reconstruction involves mapping the vector back to float numbers. A similar process can be applied to principal coefficients, although the decrease in storage is less significant.

Implementation details for obtaining the numbers in Tab. 2. We run the same test 100 times for each method without other ongoing programs, and then compute the mean as the result of the time needed to process a single batch. We perform partial fine-tuning on the final block of the transformer network. We opt for a batch size of 32, a learning rate of 5e-6, a weight decay of 0.05, and conduct fine-tuning over 10 epochs.

S-2.2 Frequency-weighted sampling method details

The idea of **Frequency Weighted Sampling** (FWS) is to assign more sampling weights for recent samples while ensuring that older samples can be selected. Specifically, when a new sample x_i is received, its sampling weight w_i is initialized to 1. After including the sample in a training batch, the weight is updated by:

$$w_i = \max(w_i * \xi, w_{\min}), \tag{S3}$$

where $\xi \in [0, 1]$ is the decay multiplier. To be clear, w_i is proportional to the likelihood that a training sample is used in the next training batch, not as a loss weight. We set a lower bound $w_{\min}$ for sampling weights, for numerical stability and to ensure relatively well-trained instances have equal chance of being sampled. When a new training example is received, a batch of size B is filled with that example and $B - 1$ other random stored examples, drawn with a w_i weighting. One training iteration is performed with that batch. The FWS strategy hastens convergence to the expectation that all training samples are eventually sampled an equal number of times, regardless of their order of appearance. ξ is not critical to performance: whether it is 0 or 1, it is equivalent to uniform sampling; any values in between adjust the degree of frequency bias—larger values lead to more repeated sampling of recently added examples. In the comparison, we set it to 0.99.

S-2.3 More implementation details

If not otherwise specified, we use the same seed for all experiments for consistency. Our dataset splits on class, data, task incremental, and flexible inference are the same as Zhu et al. [64], for fair comparison. The offline AnytimeCL model is trained with 10 epochs for each stage.

S-3 Additional ablation experiments

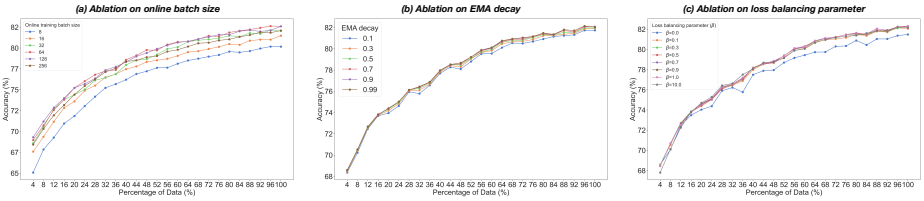

Fig. S1: Additional ablation results. Best viewed in color and zoomed-in. All experiments in these plots are conducted under the union data incremental setting. Our default selected options are 32 as the online training batch size in (a), 0.99 as the EMA decay for accuracy estimation in (b), 0.1 as the loss balancing parameter in (c).

S-3.1 Ablation on the online training batch size

For our online training method, assume the batch size is B, the total computation allowed for training new samples equals to processing B epochs of new samples. Therefore, our online learning method can fit to different computation budgets by using different online learning batch sizes. We evaluate the impact of batch size on union data incremental in Fig. S1 (a). For this comparison, we keep the ratio of learning rate and batch size constant. When the batch size is smaller than 32, the corresponding accuracy, especially in later portions of data received, is obviously lower than other batch sizes. When the batch size is 64 or 128, the performance can be better than using 32. However, the performance of batch size 256 is relatively worse than 32. This shows that given larger computation budget (*i.e.*, increasing batch size), performance can be improved though the gain diminishes quickly.

S-3.2 Ablation on the EMA decay for the accuracy estimation of OCW

As can be seen in Fig. S1 (b), our method is robust against different EMA decays but generally larger EAM decay leads to better result. As a result, we choose 0.99 as its default value throughout all experiments.

S-3.3 Ablation on the loss balancing parameter β

In Sec. 4.4 of the main paper, we already showed using the "other" logit regularization enhances the performance. Here in Fig. S1 (c), we additionally test the performance of various loss balancing weights. The results validate including the regularization term is important but the exact value of the loss balancing parameter is not critical to performance.

S-3.4 Ablation on the number of components for per-instance PCA

As default, we use only 5 components for per-instance PCA in Tab. 2. Here, we additionally present results using 3, 10, and 20 components in Tab. S1. Using 5 components maintains good accuracy after applying our proposed techniques (+CLS-weight and int-quantize). It is possible to compress further by using 3 components but the average accuracy also decreases. The gains of introducing more components than 10 are not significant.

No. components	Per-instance		+CLS-weight		+int-quantize	
	FT Acc.	KB/sample	FT Acc.	KB/sample	FT Acc.	KB/sample
3	75.2 ± 0.23	12.9	77.4 ± 0.26	12.9	77.4 ± 0.22	3.7
5	74.7 ± 0.34	19.4	77.7 ± 0.19	19.4	77.6 ± 0.16	5.3
10	77.4 ± 0.17	35.8	78.1 ± 0.14	35.8	77.7 ± 0.25	9.4
20	77.7 ± 0.11	68.5	77.8 ± 0.35	68.5	77.7 ± 0.20	17.7

Table S1: Fine-tuning accuracy for CIFAR100 when choosing different number of components. FT accuracy is obtained from averaging six runs with different seeds. Numbers after $\pm$ denote the standard deviations of FT accuracies across six runs.

S-3.5 Ablation on the per-instance PCA compression methods under union data incremental

To conduct the comparison to using full features on the union data incremental scenario, for the per-PCA compression methods, we load all cached contents to RAM and recover the intermediate features on the fly. Using full features requires extensive I/O operations to load stored features from disk during training, though it no reconstruction is needed. As presented in Fig. S2, from comparing per-instance PCA and +CLS-weight, reweighting the patch tokens using their similarities to the CLS token dramatically improves performance. The int quantization technique compresses the data further without compromising on performance. Both +CLS-weight and +int-quantization are close to the full features method in every stage, showing the effectiveness of our proposed compression methods.

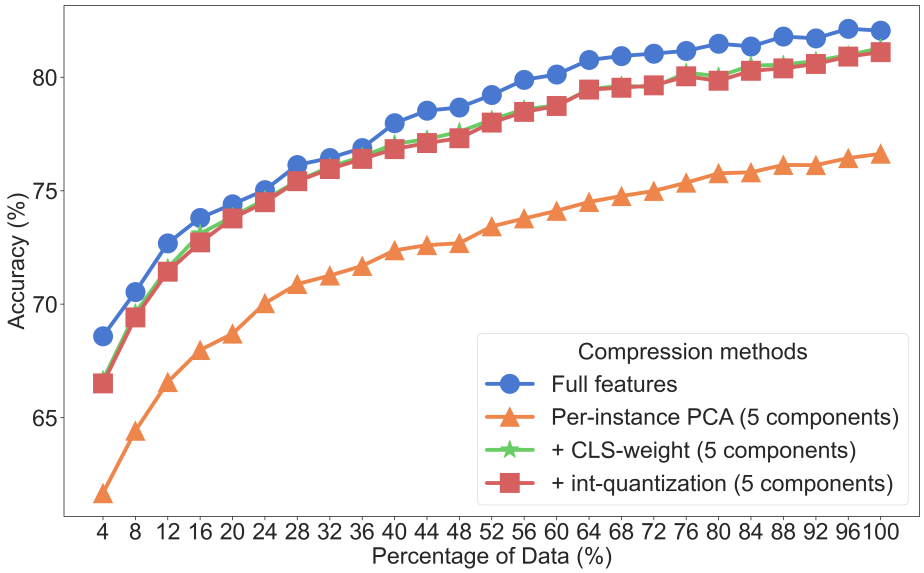

Fig. S2: Comparison of different per-instance PCA compression methods under the union data incremental scenario.

S-4 Additional results to the main draft

S-4.1 DINOv2 [31] vs. CLIP [38] tuned model on class, data, and union data incremental

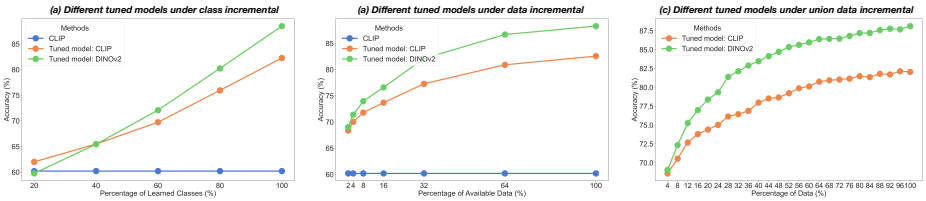

Fig. S3: Comparison between DINOv2 [31] and CLIP [38] as tuned model on class (a), data (b), and union data incremental scenarios (c).

Fig. S3 shows the comparison. Clearly using DINOv2 as the tuned model has better results eventually in all cases. It is interesting that DINOv2 as the tuned model shows superiority over CLIP in the very beginning of data incremental scenarios in Fig. S3 (b) & (c), unlike the class incremental case in (a). This result may indicate that DINOv2 is more helpful in few-shot learning scenarios.

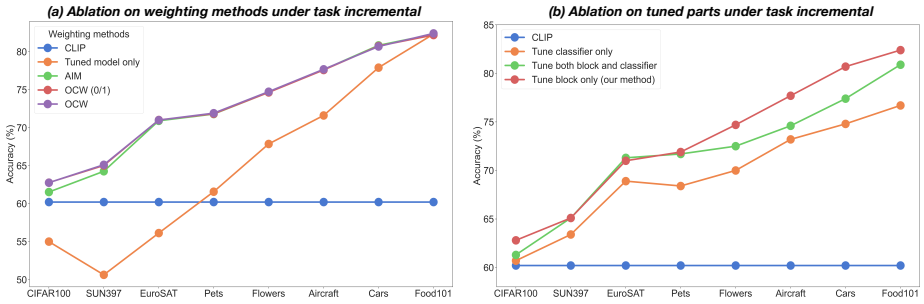

Fig. S4: Supplemental results to existing ablations in the main draft. In (a), OCW (0/1) is hard to be observed since OCW overlaps with it.

S-4.2 Weighting method ablation on task incremental

Fig. S4 (a) shows the results of different weighting methods on the task incremental scenario. This scenario cannot fully demonstrate the effectiveness of weighting methods for the similar reason stated in Sec. 4.2.

S-4.3 Tuned parts ablation on task incremental

Our proposed method only tunes the last transformer block and freezes the label encoder. This leads to better target task performance than replacing the label encoder with a tuned classifier, as shown in Fig. S4 (b). Training only the classifier layer underperforms because it cannot tune the features. When fine tuning the last transformer block, keeping the label encoder fixed improves performance, perhaps because it provides a regularization on the feature tuning.

S-4.4 Detailed results under the MTIL task incremental learning benchmark [63]

We also present the detailed accuracies of AnytimeCL online and offline in Tab. S2 and Tab. S3, respectively. Our internal tests revealed that hyperparameter tuning, such as adjusting the learning rate for each task, can enhance results, as revealed in ZSCL's released code [63]. However, we tend to avoid hyperparameter tuning of different tasks since selected hyperparameters do not guarantee a better performance when the data distribution in an online data stream shifts.

	Aircraft	Caltech101	CIFAR100	DTD	EuroSAT	Flowers	Food	MNIST	OxfordPet	Cars	SUN397	
Transfer		87.9	68.2	45.3	54.6	71.4	88.9	59.4	89.1	64.6	64.1	69.4
Aircraft	44.85	87.90	68.22	45.32	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
Caltech101	50.50	96.60	68.22	45.32	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
CIFAR100	52.45	96.89	82.23	45.32	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
DTD	52.42	96.66	83.03	69.63	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
EuroSAT	52.78	96.77	83.57	75.64	94.46	71.43	88.86	59.45	89.07	64.61	64.05	
Flowers	53.59	96.83	83.52	74.95	95.59	87.84	88.86	59.45	89.07	64.61	64.05	
Food	54.04	96.77	83.6	75.11	96.63	92.83	91.36	59.45	89.07	64.61	64.05	
MNIST	54.40	96.49	83.77	75.32	96.19	93.23	91.60	98.51	89.07	64.61	64.05	
OxfordPet	55.12	96.43	83.54	75.37	96.83	92.97	92.22	98.76	91.63	64.61	64.05	
Cars	53.44	96.60	83.68	74.73	96.63	92.94	92.10	98.58	92.75	83.48	64.05	
SUN397	53.11	96.37	83.27	73.51	95.93	92.88	92.04	98.36	93.16	85.77	79.67	85.8
Avg.	52.4	95.8	80.6	66.4	81.0	82.7	90.2	73.7	90.0	68.2	65.5	77.0

Table S2: Accuracy (%) of AnytimeCL Online on the MTIL benchmark with order-I. Each row represents the performance on every dataset of the model trained after the corresponding task. Transfer, Avg., and Last metrics are shown in color. We follow the same table arrangement as in ZSCL [63].

	Aircraft	Caltech101	CIFAR100	DTD	EuroSAT	Flowers	Food	MNIST	OxfordPet	Cars	SUN397	
Transfer		87.9	68.2	45.3	54.6	71.4	88.9	59.4	89.1	64.6	64.1	69.4
Aircraft	36.87	87.90	68.22	45.32	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
Caltech101	45.48	97.00	68.22	45.32	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
CIFAR100	49.68	96.72	84.46	45.32	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
DTD	51.28	96.77	84.14	73.83	54.61	71.43	88.86	59.45	89.07	64.61	64.05	
EuroSAT	52.18	96.71	84.48	74.79	97.37	71.43	88.86	59.45	89.07	64.61	64.05	
Flowers	53.02	96.89	84.45	75.16	97.59	91.67	88.86	59.45	89.07	64.61	64.05	
Food	52.39	96.95	84.20	74.41	97.54	91.72	92.43	59.45	89.07	64.61	64.05	
MNIST	53.17	96.66	84.3	74.73	97.59	91.90	92.48	98.99	89.07	64.61	64.05	
OxfordPet	53.92	96.83	84.45	74.31	97.52	91.88	92.35	99.16	93.92	64.61	64.05	
Cars	53.80	97.06	84.23	75.05	97.50	91.54	92.41	99.20	94.11	85.86	64.05	
SUN397	52.48	96.72	83.83	73.94	97.33	91.56	92.23	99.14	93.73	85.84	81.50	86.2
Avg.	50.4	96.0	81.3	66.6	81.9	82.5	90.5	73.9	90.4	68.5	65.6	77.0

Table S3: Accuracy (%) of AnytimeCL Offline on the MTIL benchmark with order-I.