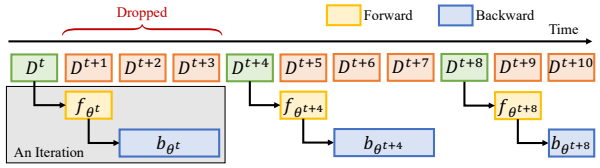
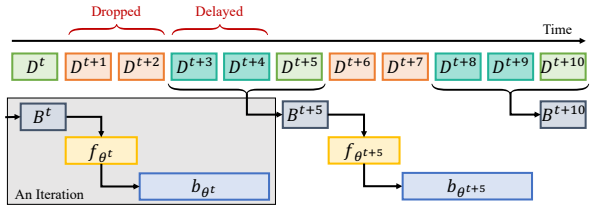
Ferret: An Efficient Online Continual Learning Framework under Varying Memory Constraints

Supplementary Material



(a) Online continual learning



(b) Mini-batch online continual learning

Figure 8. (a) A typical online continual learning framework that trains the model as soon as the data arrives. (b) A mini-batch of data is processed for each iteration, reducing dropped data but introducing delayed data.

8. Additional Related Work

Online Continual Learning: OCL is crucial in ML for adapting to continuously evolving data, primarily addressing catastrophic forgetting [2, 3, 12, 47] and concept drift in data streams under resource constraints [11, 26, 48]. Some methods [29, 70] reduce latency by immediately processing new data, as Fig. 8a illustrated, but this often discards valuable information and hampers comprehensive learning. To preserve data, several approaches [46, 54, 81, 88, 90] buffer it for later batch-training, as Fig. 8b illustrated, which, while increasing latency and computational costs, especially with complex algorithms like coreset selection, seeks to avoid data skipping.

Distributed and parallel ML frameworks: Distributed and parallel ML frameworks are essential for processing large data sets with minimal latency [55, 64, 67, 72, 73, 89]. For instance, Pytorch [64], Megatron-LM [73], and DeepSpeed [67] use GPipe [38] and PipeDream-Flush [59] to enable concurrent processing across model segments, enhancing real-time data management. Horovod [72] streamlines synchronized distributed training across TensorFlow [1], Keras [15], and PyTorch, optimizing ML training on vast computing clusters. Conversely, Ray [55] supports large-scale asynchronous distributed training with custom gradient compensation algorithms [85].

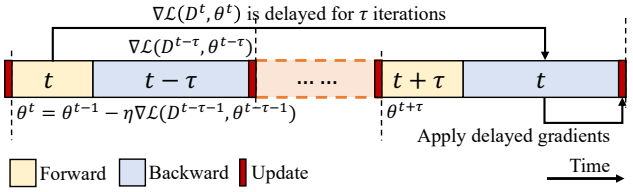


Figure 9. Due to asynchronous model updating, $\nabla\mathcal{L}(D^t, \theta^t)$ is obtained when θ^t has been updated to $\theta^{t+\tau}$

Parallelization optimizations: Several strategies beyond data and model parallelism have been developed to optimize parallelization in ML. Pipeline parallelism has seen notable advancements with the introduction of strategies like DAPPLE [24], Pipedream [58], Zero-Bubble [66], and Hanayo [49], significantly boosting training efficiency. Additionally, during asynchronous parallelization, the model will be inevitably updated by stale gradients, as Fig. 9 illustrated, leading to low data management efficiency and performance degradation. Strategies to counteract stale gradients in parallel processing include reducing step sizes for outdated gradients [7, 33, 41] or using higher-order information [85, 87].

9. Notations

Let us assume the model θ in the hypothesis space Θ has $\hat{L}$ layers, with $\hat{t}_i^f$, $\hat{t}_i^b$, $|\hat{w}_i|$, $|\hat{a}_i|$ denoting the time consumed for the forward and backward pass, the size of the model parameters and the output activations of the i-th layer of the model, respectively. The model is then partitioned into $P = |L| - 1$ pipeline stages, where L represents the model partition scheme. The j-th stage encompasses the L_j-th to the $(L_{j+1} - 1)$-th layer of the model, with the size of model parameters $|w_j| = \sum_{i=L_j}^{L_{j+1}-1} |\hat{w}_i|$ and the size of activations $|a_j| = \sum_{i=L_j}^{L_{j+1}-1} |\hat{a}_i|$. Thus, the time consumed for the forward and backward pass of a stage in pipeline parallelism is $t^f = \max_j\{\sum_{i=L_j}^{L_{j+1}-1} |\hat{t}_i^f|\}$ and $t^b = \max_j\{\sum_{i=L_j}^{L_{j+1}-1} |\hat{t}_i^b|\}$, respectively. Moreover, after partitioning, a pipeline configuration C is determined based on the interval between data arrivals t^d that contains N worker configurations, where the n-th worker configuration specifies the processing delay c_n^d, the activation recomputation indicator c_n^r, and the gradient accumulation and back-propagation omission steps of its j-th stage $c_{n,j}^a$, $c_{n,j}^o$, respectively.

Table 5. The glossary of notations

Notation	Implication
$\theta \in \Theta$	Model parameters
$\hat{L}$	Number of layers in θ
$\hat{t}_i^f$	Time consumed for the i-th layer's forward pass
$\lvert\hat{w}_i\rvert$	Number of parameters in the i-th layer's
$\lvert\hat{a}_i\rvert$	Number of parameters for the i-th layer's output activations
L	Partition scheme, the j-th partition has $[L_j, L_{j+1})$ layers
P	Number of pipeline stages $\lvert L\rvert - 1$
$\lvert w_j\rvert$	Number of parameters in the j-th stage $\sum_{i=L_j}^{L_{j+1}-1} \lvert\hat{w}_i\rvert$
$\lvert a_j\rvert$	Number of activations in the j-th stage $\sum_{i=L_j}^{L_{j+1}-1} \lvert\hat{a}_i\rvert$
t^f	Time consumed for the forward pass of a stage $\max_j\{\sum_{i=L_j}^{L_{j+1}-1} \lvert\hat{t}_i^f\rvert\}$
t^b	Time consumed for the backward pass of a stage $\max_j\{\sum_{i=L_j}^{L_{j+1}-1} \lvert\hat{t}_i^b\rvert\}$
t^d	Interval between data arrivals
C	Pipeline configuration
c_n^d	Processing delay of the n-th worker
c_n^r	Actication recomputation indicator of the n-th worker
$c_{n,j}^a$	Number of gradient accumulation steps of the j-th stage in the n-th worker
$c_{n,j}^o$	Number of back-propagation omission steps of the j-th stage in the n-th worker
D^t	Data arrived at timestamp t
V_{D^t}	Data value of D^t
$\mathcal{R}_{\mathcal{A}}^T$	Adaptation rate of a OCL framework $\mathcal{A}$ for $t \in [0, T]$
$\mathcal{M}_{\mathcal{A}}$	Memory footprint of $\mathcal{A}$
$oacc_{\mathcal{A}}(t)$	Online accuracy of $\mathcal{A}$ at timestamp t
$tacc_{\mathcal{A}}(t)$	Test accuracy of $\mathcal{A}$ at timestamp t
$agm_{\mathcal{B}}(\mathcal{A}, t)$	Online Accuracy Gain per unit of Memory of $\mathcal{A}$ over $\mathcal{B}$ at timestamp t
$tagm_{\mathcal{B}}(\mathcal{A}, t)$	Test Accuracy Gain per unit of Memory of $\mathcal{A}$ over $\mathcal{B}$ at timestamp t

10. Adaptation Rate and Memory Reduction for S1-S4

In Sec. 5.2.1, **T1-T4** are progressively employed to reduce $\mathcal{R}_F^T$ and $\mathcal{M}_F$.

S1. Deploy T1 for all workers: By setting $c_n^r = 1$ for all workers, the data processing time increases. Specifically, for the n-th worker, setting $c_n^r = 1$ will respectively reduce $\mathcal{R}_F^T$ and $\mathcal{M}_F$ by:

$$\Delta\mathcal{R}_F^T = \sum_{i=0}^{P-1} \frac{|w_i|}{\sum_{j=0}^{P-1}(|w_j|)} \frac{1}{c_{n,i}^a} \sum_{j=0}^{c_{n,i}^a - 1} (B_{i,j} - C_{i,j}),$$

$$\text{where } B_{i,j} = \frac{e^{-c((2P+2j-i)t^f + (P-i+j)t^b)} V_D}{LCM(\{c_{n,k}^o + 1 | k \in [i, P-1]\})(2t^f + t^b)},$$

$$C_{i,j} = \frac{e^{-c((P+j)t^f + (P-i+j)t^b)} V_D}{LCM(\{c_{n,k}^o + 1 | k \in [i, P-1]\})(t^f + t^b)},$$

$$\Delta\mathcal{M}_F = -\sum_{i=0}^{P-1} (1 + \lceil \frac{P-i-1}{c_{n,i}^a} \rceil - c_{n,i}^o) \sum_{l=L_i+1}^{L_{i+1}-1} |\hat{a}_l|. \quad (19)$$

S2. Deploy T2 for the j-th stage in the n-th worker: If $c_{n,j}^o = 0$, increasing $c_{n,j}^a$ by $\Delta c_{n,j}^a = \lceil \frac{P-j-1}{\lceil (P-j-1)/c_{n,j}^a \rceil - 1} \rceil - c_{n,j}^a$ will lead to a reduced frequency of model parameter updates. Here, the value of $\Delta c_{n,j}^a$ is determined to prevent $\Delta_{c_{n,j}^a \to c_{n,j}^a + 1} \mathcal{M}_F = 0$ due to the ceiling function. Consequently, $\mathcal{R}_F^T$ and $\mathcal{M}_F$ will be respectively decreased by:

$$\Delta\mathcal{R}_F^T = \frac{|w_j|}{\sum_{k=0}^{P-1}(|w_k|)} \left(\frac{\sum_{k=c_{n,j}^a}^{c_{n,j}^a + \Delta c_{n,j}^a - 1} A_{j,k}}{c_{n,j}^a + \Delta c_{n,j}^a} - \frac{\Delta c_{n,j}^a \sum_{k=0}^{c_{n,j}^a - 1} A_{j,k}}{(c_{n,j}^a + \Delta c_{n,j}^a) c_{n,j}^a} \right),$$

$$\Delta\mathcal{M}_F = -(|w_j| + |a_j| - c_n^r \sum_{l=L_j+1}^{L_{j+1}-1} |\hat{a}_l|). \quad (20)$$

S3. Deploy T3 For the j-th stage in the n-th worker: If $\Delta c_{n,j}^a = +\infty$, setting $c_{n,j}^a = 1$ and $c_{n,j}^o = P - 1 - j$ will completely eliminate the need for the j-th stage in the n-th worker to store additional model parameters by bypassing any backward pass that requires previous model parameters. Consequently, $\mathcal{R}_F^T$ and $\mathcal{M}_F$ will be respectively reduced by:

$$\Delta\mathcal{R}_F^T = \sum_{i=0}^{P-1-j} \frac{|w_i|}{\sum_{k=0}^{P-1}(|w_j|)} \frac{\sum_{j=0}^{c_{n,i}^a - 1} (E_{i,j} - A_{i,j})}{c_{n,i}^a}, \text{where } E_{i,j} = \frac{e^{-c((P+j)t^f + (P-i+j)t^b + c_n^r (P-i+j)t^f)} V_D}{LCM(LCM(\{c_{n,k}^o + 1 | k \in [i, P-1]\}), P-j)(t^f + t^b + c_n^r t^f)},$$

$$\Delta\mathcal{M}_F = -\lceil \frac{P-j-1}{c_{n,j}^a} \rceil (|w_j| + |a_j| - c_n^r \sum_{l=L_j+1}^{L_{j+1}-1} |\hat{a}_l|). \quad (21)$$

S4. Deploy T4 for the n-th worker: If $c_{n,j}^o \neq 0$ for all $j \in [0, p-1)$, shutting down the n-th worker will lead to a respective decrease in $\mathcal{R}_F^T$ and $\mathcal{M}_F$ by:

$$\Delta\mathcal{R}_F^T = -\sum_{i=0}^{P-1} \frac{|w_i|}{\sum_{j=0}^{P-1}(|w_j|)} \frac{1}{c_{n,i}^a} \sum_{j=0}^{c_{n,i}^a - 1} A_{i,j},$$

$$\text{where } A_{i,j} = \frac{e^{-c((P+j)t^f + (P-i+j)t^b + c_n^r (P-i+j)t^f)} V_D}{LCM(\{c_{n,k}^o + 1 | k \in [i, P-1]\})(t^f + t^b + c_n^r t^f)},$$

$$\Delta\mathcal{M}_F = -\sum_{i=0}^{P-1} (1 + \lceil \frac{P-i-1}{c_{n,i}^a} \rceil - c_{n,i}^o)(|w_i| + |a_i| - c_n^r \sum_{l=L_i+1}^{L_{i+1}-1} |\hat{a}_l|). \quad (22)$$

11. Algorithm Details

Algorithm 1 initially updates λ for lower approximation errors (Line 3-7). Then, the algorithm iteratively reduces the staleness of gradients step by step, given the hyper-parameter λ (Line 8-9). Afterward, the model parame-

Algorithm 1: Iterative Gradients Compensation with λ

Input: $\nabla\mathcal{L}(D^{t-1};\theta^{t-1})$; $\theta^{t-1},\theta^{t},\ldots,\theta^{t+\tau-1}$; α; η and η_λ; λ^0;
Output: $\theta^{t+\tau}$;

1 **Function** ***compensate*** ($\nabla\mathcal{L}(D^{t-1};\theta^{t-1})$, θ^{t-1}, θ^{t}, $\ldots$, $\theta^{t+\tau-1}$)
// initialization
2 $\lambda=\lambda^0$; $v_r=v_a=0$
3 **if** $\eta_\lambda>0$ **then**
// update λ
4 $\Delta v_r=(1-\alpha)(\nabla\mathcal{L}(D^{t-1};\theta^{t-1})-v_r)$
5 $\lambda=\lambda-\eta_\lambda\nabla_\lambda||\Delta v_r-\lambda v_a||^2$
6 $v_r=\alpha v_r+(1-\alpha)\nabla\mathcal{L}(D^{t-1};\theta^{t-1})$
7 $v_a=\alpha v_a+(1-\alpha)(\nabla\mathcal{L}(D^{t-1};\theta^{t-1})\odot\nabla\mathcal{L}(D^{t-1};\theta^{t-1})\odot(\theta^{t}-\theta^{t-1}))$
// iterative approximation
8 **forall** i *in* $0,...,\tau-1$ **do**
9 $\nabla\mathcal{L}(D^{t-1};\theta^{t+i})=A_{\mathcal{I}}(\nabla\mathcal{L}(D^{t-1};\theta^{t+i-1}),\theta^{t+i},\theta^{t+i-1})$
// model update
10 $\theta^{t+k}=\theta^{t+\tau-1}-\eta\nabla\mathcal{L}(D^{t-1};\theta^{t+\tau-1})$
11 **return** $\theta^{t+\tau}$

Algorithm 2: Iterative Configuration Search

Input: t^d; t^f; t^b; L; M;
Output: $C=\{c_n^r,c_n^d,c_{n,j}^a,c_{n,j}^o\}$; $\mathcal{R}_F^T$;

1 **Function** ***itersearch*** (t^d, t^f+t^b, c^r, L, M)
// initialization
2 $N=\lceil(t^f+t^b+c^rt^f)/t^d\rceil$
3 $c_n^d=n$; $c_n^r=c^r$; $c_{n,j}^a=1$; $c_{n,j}^o=0$
// iterative search
4 Calculate $\mathcal{R}_F^T$ and $\mathcal{M}_F$
5 **while** $\mathcal{M}_F>M$ **do**
6 **forall** n *in* $0,\ldots,N-1$ **do**
7 **forall** p *in all* {***S2, S3, S4***} **do**
8 Calculate $\Delta_p\mathcal{M}_F/\Delta_p\mathcal{R}_F^T$
9 $p^*=\arg\max_p(\Delta_p\mathcal{M}_F/\Delta_p\mathcal{R}_F^T)$
10 Update $c_n^r,c_n^d,c_{n,j}^a,c_{n,j}^o,\mathcal{R}_F^T,\mathcal{M}_F$ based on p^*
11 **return** $C=\{c_n^r,c_n^d,c_{n,j}^a,c_{n,j}^o\},\mathcal{R}_F^T$
12 **Function** ***search*** (t^d, t^f+t^b, L, M)
13 $C_0,\mathcal{R}_0=itersearch(t^d,t^f+t^b,0,L,M)$
14 $C_1,\mathcal{R}_1=itersearch(t^d,t^f+t^b,1,L,M)$
15 **return** $C_i,\mathcal{R}_i$ *where* $i=\arg\max_i\{\mathcal{R}_i\}$

ter $\theta^{t+\tau-1}$ is updated by the delay-compensated gradient $\nabla\mathcal{L}(D^{t-1};\theta^{t+\tau-1})$ using gradient descent algorithm (Line 10).

Algorithm 2 begins by initializing N, c_n^d, c_n^r, $c_{n,j}^a$ and $c_{n,j}^o$ using the given parameters P, t^d, t^f, c^r and t^b (Lines 2-3). Then, the initial $\mathcal{R}_F^T$ and $\mathcal{M}_F$ are computed using Eq. 4 (Line 4). The algorithm then repeatedly evaluate $\Delta_p\mathcal{M}_F$ and $\Delta_p\mathcal{R}_F^T$ based on the aforementioned deployments {**S2**, **S3**, **S4**}, applying the one that maximizes $\Delta_p\mathcal{M}_F/\Delta_p\mathcal{R}_F^T$ until $\mathcal{M}_F\leq M$ (Lines 5-10). Finally, **S1** is evaluated separately in Lines 13-14 since it is applied to all workers and impacts t^b and N.

Algorithm 3 starts by listing all possible t^c for a stage, where $profile(\cdot)$ measures the number of layers and their respective statistics. (Line 3-8). For each t^c, a model partition scheme L is then constructed (Line 11-16). Finally, the optimal C and $\mathcal{R}$ under the formed L are retrieved using the $search(\cdot)$ in Alg. 2 (Line 17). By comparing $\mathcal{R}$ of different L, L^* and C^* can be obtained (Line 18-19).

12. Implementation Details

Datasets: To align with our computational resources, images in Tiny-ImageNet [45] and CORe50 [50] are resized to 32×32, while images in CLEAR10 and CLEAR100 [48] are resized to 224×224. Additionally, split datasets (*i.e.*, Split-MNIST, Split-CIFAR10, etc.) are partitioned into 5 tasks to simulate a class-incremental setting [9, 20, 46]. CORe50-iid is a shuffled version of the CORe50 dataset.

The rationale behind our evaluation metrics: To comprehensively evaluate both performance and memory footprint of given OCL frameworks $\mathcal{A}$ and $\mathcal{B}$, it is natural to consider the ratio of their differences:

$$metric_c=\frac{metric_{\mathcal{A}}-metric_{\mathcal{B}}}{\log(M_{\mathcal{A}})-\log(M_{\mathcal{B}})}, \tag{23}$$

where $metric_c$ is the comprehensive metric that shows the performance improvement per memory increment, and $metric_{\mathcal{A}}$ and $metric_{\mathcal{B}}$ are the arbitrary performance metrics of $\mathcal{A}$ and $\mathcal{B}$, respectively. It is important to note that the logarithm function is employed to account for the diminishing returns associated with memory increments. Then, Eq. 23 can be further represented as:

$$metric_c\sim\frac{metric_{\mathcal{A}}-metric_{\mathcal{B}}}{\log(M_{\mathcal{A}}/M_{\mathcal{B}})} \tag{24}$$

$$\sim\frac{\exp(metric_{\mathcal{A}}-metric_{\mathcal{B}})}{M_{\mathcal{A}}/M_{\mathcal{B}}} \tag{25}$$

$$\sim\log(\frac{\exp(metric_{\mathcal{A}}-metric_{\mathcal{B}})}{M_{\mathcal{A}}/M_{\mathcal{B}}}), \tag{26}$$

which is the same form as Eq. 17 and Eq. 18.

Hyper-parameters: All experiments are conducted on a server with 32 Intel(R) Xeon(R) CPU E5-2620 v42.10GHz

Algorithm 3: Brute-force planning

Input: t^d; θ; M; $profile(\cdot)$; $search(\cdot)$;
Output: L^* and C^* that maximizes Eq. 13;

Function *plan* (t^d, θ, M)
  $\hat{L}, \hat{t}_i^f, \hat{t}_i^b, |\hat{w}_i|, |\hat{a}_i| = profile(\theta); S = \varnothing$
  `// get all consumed time for a stage`
  forall k *in* $0, \ldots, \hat{L} - 1$ **do**
    forall i *in* $0, \ldots, \hat{L} - 1 - k$ **do**
      $t^c = \sum_{j=i}^{i+k} (\hat{t}_j^f + \hat{t}_j^f)$
      if $(t^c < \max_{j \in [0, L-1]} (\hat{t}_j^f + \hat{t}_j^b)$ **then**
        $S = S \cup t^c$
  $S = sorted(S)$
  `// Search for the optimal scheme`
  $L^* = []; t^{c*} = \mathcal{R}^* = 0; C^* = \varnothing$
  forall t^c *in* S **do**
    $L = [0]; t^{sum} = 0$
    forall i *in* $0, \ldots, \hat{L} - 1$ **do**
      $t^{sum} = t^{sum} + \hat{t}_i^f + \hat{t}_i^b$
      if $t^{sum} > t^c$ **then**
        $L.append(i); t^{sum} = \hat{t}_i^f + \hat{t}_i^b$
    $L.append(\hat{L})$
    $C, \mathcal{R} = search(t^d, t^c, L, M)$
    if $\mathcal{R} > \mathcal{R}^*$ **then**
      $\mathcal{R}^* = \mathcal{R}; t^{c*} = t^c; L^* = L; C^* = C;$
  return L^*, C^*

CPUs, 8 NVIDIA TITAN Xp GPUs and 64 GB memory. The learning rate is 1e-3, the interval between data arrivals is set to maximal time consumed for the forward pass of a layer in the model, *i.e.*, $t^d = \max_i\{\hat{t}_i^f\}$, and the replay buffer size for all OCL algorithms is 5e3. For Iter-Fisher, λ = 0.2, μ = 2e-6, while for the other compensation algorithms, λ is manually tuned for each dataset. Additionally, L^* and C^* are pre-determined and shared for all pipeline parallelism strategies using tuned c and M for each dataset.

13. Additional Evaluation Results

In this section, we present additional results for the proposed Ferret and the baseline methods, employing the standard Online Accuracy and Test Accuracy metrics. These metrics are widely recognized as the standard measure in the field of OCL. Notably, The following results do not account for the memory footprint during training, which is the main reason why they are not included in the main paper.

Table 7 shows the online accuracy of different OCL frameworks on various datasets. Thanks to pipeline parallelism, the model parameters can be updated more frequently, leading to a higher online accuracy even for $\text{Ferret}_{\text{M}-}$. $\text{Ferret}_{\text{M}+}$, which has the highest memory footprint, achieves the best online accuracy on all datasets. The results demonstrate that Ferret can effectively utilize memory resources to improve the online accuracy of the model.

Table 8 shows the online accuracy and test accuracy of different integrated OCL algorithms on the CORe50/ConvNet dataset. From the table, we can see that both ER, MIR, LWF, and MAS can effectively mitigate catastrophic forgetting, achieving better test accuracy while maintaining comparable online accuracy. When applying these algorithms to different OCL frameworks, $\text{Ferret}_{\text{M}+}$ consistently outperforms the other OCL frameworks by a large margin, demonstrating the effectiveness of Ferret in leveraging memory resources to improve the performance of integrated OCL algorithms.

Fig. 10 and Fig. 11 expand results in Fig. 4 and Fig. 6 to show the memory consumption of different stream learning algorithms and the relationship between online accuracy and memory consumption of different pipeline parallelism strategies, respectively.

Table 6. Statistics of raw datasets.

Dataset	# Stream Data	# Features	# Classes
MNIST [22]	60,000	784	10
FMNIST [79]	60,000	784	10
EMNIST [16]	697,932	784	62
CIFAR10 [43]	50,000	3,072	10
CIFAR100 [43]	50,000	3,072	100
SVHN [60]	604,388	3,072	10
Tiny-ImageNet [45]	100,000	12,288	200
CORe50 [50]	119,894	49,152	50
CORe50-iid	119,894	49,152	50
Split-MNIST	60,000	784	10
Split-FMNIST	60,000	784	10
Split-CIFAR10	50,000	3,072	10
Split-CIFAR100	50,000	3,072	100
Split-SVHN	73,257	3,072	10
Split-Tiny-ImageNet	100,000	12,288	200
Covertype [8]	464,809	54	7
CLEAR10 [48]	330,000	562,500	11
CLEAR100 [48]	1,209,197	562,500	101

14. Limitations

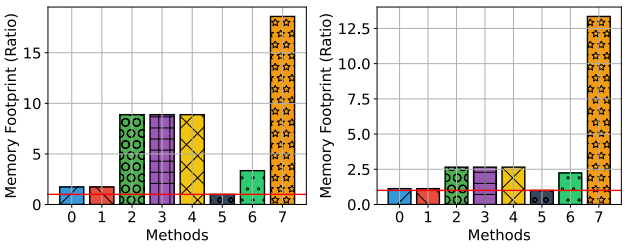
To reduce the staleness of gradients, the proposed Ferret utilizes Taylor series expansion to approximate the gradient at the current time step. This compensation introduces an additional hyper-parameter λ. However, the optimal λ may vary across different datasets and models, which may require manual tuning. To automate this process, λ can be optimized in real-time under the mild assumption that the distributions of $\mathbb{E}_k D^k$ and $\mathbb{E}_k D^{k+1}$ are similar. This as-

Table 7. Online Accuracy of different algorithms. "M-", "M", "M+" refer to the ferret method with minimal, medium and maximal memory footprint, respectively.

Setting	Oracle	1-Skip	Random-N	Last-N	Camel	Ferret$_{\mathrm{M-}}$	Ferret$_{\mathrm{M}}$	Ferret$_{\mathrm{M+}}$
MNIST/MNISTNet	81.14$_{\pm 1.43}$	18.24$_{\pm 3.05}$	18.48$_{\pm 2.65}$	18.87$_{\pm 3.17}$	17.82$_{\pm 2.7}$	30.16$_{\pm 4.57}$	56.25$_{\pm 3.69}$	**80.98**$_{\pm 1.44}$
FMNIST/MNISTNet	65.94$_{\pm 0.64}$	21.39$_{\pm 2.71}$	21.9$_{\pm 1.66}$	22.02$_{\pm 1.57}$	21.22$_{\pm 2.12}$	34.76$_{\pm 2.89}$	51.19$_{\pm 1.72}$	**65.78**$_{\pm 0.66}$
EMNIST/MNISTNet	75.91$_{\pm 0.14}$	45.98$_{\pm 1.23}$	51.64$_{\pm 1.14}$	51.82$_{\pm 1.15}$	50.73$_{\pm 1}$	55.33$_{\pm 1.04}$	66.8$_{\pm 0.31}$	**75.9**$_{\pm 0.15}$
Cifar10/ConvNet	51.84$_{\pm 0.04}$	27.5$_{\pm 0.19}$	40.09$_{\pm 0.26}$	40.24$_{\pm 0.21}$	40.06$_{\pm 0.11}$	34.25$_{\pm 0.56}$	42.46$_{\pm 0.18}$	**51.69**$_{\pm 0.36}$
Cifar100/ConvNet	14.56$_{\pm 0.03}$	2.49$_{\pm 0.01}$	5.81$_{\pm 0.16}$	5.92$_{\pm 0.13}$	5.75$_{\pm 0.17}$	5.66$_{\pm 0.08}$	9.15$_{\pm 0.06}$	**14.89**$_{\pm 0.12}$
SVHN/ConvNet	81.59$_{\pm 0.04}$	46.11$_{\pm 0.52}$	64.07$_{\pm 0.68}$	64.52$_{\pm 0.61}$	64.86$_{\pm 0.73}$	56.98$_{\pm 0.75}$	73.29$_{\pm 0.25}$	**81.6**$_{\pm 0.2}$
TinyImagenet/ConvNet	5.65$_{\pm 0.19}$	0.75$_{\pm 0.02}$	1.6$_{\pm 0.12}$	1.63$_{\pm 0.1}$	1.62$_{\pm 0.13}$	1.51$_{\pm 0.08}$	2.67$_{\pm 0.07}$	**5.61**$_{\pm 0.17}$
CORe50/ConvNet	81.59$_{\pm 0.12}$	21.69$_{\pm 1.01}$	51.25$_{\pm 0.54}$	51.59$_{\pm 0.66}$	48.81$_{\pm 0.74}$	42.05$_{\pm 0.97}$	63.68$_{\pm 0.56}$	**80.46**$_{\pm 0.22}$
CORe50-iid/ConvNet	63.8$_{\pm 0.34}$	19.49$_{\pm 6.82}$	27.74$_{\pm 6.39}$	34.34$_{\pm 0.67}$	33.14$_{\pm 1.24}$	27.11$_{\pm 1.09}$	44.89$_{\pm 0.66}$	**63.23**$_{\pm 0.34}$
SplitMNIST/MNISTNet	94.96$_{\pm 0.08}$	53.04$_{\pm 1.82}$	59.66$_{\pm 2.7}$	59.72$_{\pm 2.99}$	61.85$_{\pm 2.73}$	66.81$_{\pm 3.5}$	87.11$_{\pm 0.61}$	**94.4**$_{\pm 0.2}$
SplitFMNIST/MNISTNet	94.98$_{\pm 0.29}$	68.92$_{\pm 3.68}$	73.67$_{\pm 4.23}$	73.57$_{\pm 4.27}$	74.66$_{\pm 4.17}$	81.14$_{\pm 2.68}$	91.08$_{\pm 0.6}$	**94.7**$_{\pm 0.29}$
SplitCifar10/ConvNet	84.1$_{\pm 0.07}$	66.84$_{\pm 0.21}$	75.6$_{\pm 0.13}$	75.73$_{\pm 0.32}$	75.76$_{\pm 0.07}$	72.89$_{\pm 0.24}$	78.64$_{\pm 0.31}$	**83.55**$_{\pm 0.09}$
SplitCifar100/ConvNet	33.52$_{\pm 0.24}$	9.33$_{\pm 0.2}$	17.32$_{\pm 0.19}$	17.44$_{\pm 0.27}$	17.17$_{\pm 0.5}$	17.01$_{\pm 0.16}$	24.1$_{\pm 0.2}$	**33.72**$_{\pm 0.14}$
SplitSVHN/ConvNet	94.83$_{\pm 0.02}$	79.89$_{\pm 0.77}$	88.31$_{\pm 0.48}$	88.32$_{\pm 0.55}$	88.29$_{\pm 0.48}$	85.58$_{\pm 0.49}$	92.06$_{\pm 0.13}$	**94.74**$_{\pm 0.05}$
SplitTinyImagenet/ConvNet	5.72$_{\pm 0.27}$	0.8$_{\pm 0.06}$	1.59$_{\pm 0.13}$	1.67$_{\pm 0.1}$	1.55$_{\pm 0.06}$	1.53$_{\pm 0.08}$	2.9$_{\pm 0.07}$	**5.66**$_{\pm 0.2}$
CLEAR10/ResNet	96.4$_{\pm 0.02}$	72.52$_{\pm 0.15}$	91.42$_{\pm 0.17}$	91.63$_{\pm 0.13}$	66.71$_{\pm 24.22}$	78.02$_{\pm 0.02}$	90.96$_{\pm 0.03}$	**96.32**$_{\pm 0.04}$
CLEAR10/MobileNet	76.89$_{\pm 0.51}$	30.02$_{\pm 0.48}$	58.75$_{\pm 0.67}$	59.22$_{\pm 0.4}$	58.88$_{\pm 0.33}$	25.36$_{\pm 0.78}$	64.4$_{\pm 0.82}$	**75.18**$_{\pm 0.39}$
CLEAR100/ResNet	89.08$_{\pm 0.06}$	39.08$_{\pm 0.96}$	74.9$_{\pm 0.16}$	75.29$_{\pm 0.1}$	73.07$_{\pm 0.11}$	56.24$_{\pm 0.06}$	75.53$_{\pm 0.31}$	**89.54**$_{\pm 0.37}$
CLEAR100/MobileNet	61$_{\pm 2.49}$	6.85$_{\pm 0.12}$	29.48$_{\pm 0.69}$	30$_{\pm 0.45}$	28.48$_{\pm 0.25}$	8.69$_{\pm 0.25}$	43.81$_{\pm 0.97}$	**60.29**$_{\pm 1.44}$
Covertype/MLP	80.9$_{\pm 0.69}$	63.25$_{\pm 0.83}$	60.59$_{\pm 1.52}$	60.66$_{\pm 1.53}$	60.57$_{\pm 1.49}$	64.94$_{\pm 0.92}$	67.63$_{\pm 0.23}$	**72.95**$_{\pm 0.17}$

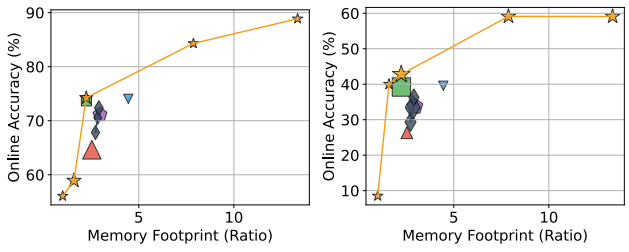
Table 8. Online Accuracy and Test Accuracy of different integrated OCL algorithms on CORe50/ConvNet. Camel has its dedicated component to mitigate catastrophic forgetting and cannot be integrated with various OCL algorithm.

	Metric	Oracle	1-Skip	Random-N	Last-N	Camel	Ferret$_{\mathrm{M-}}$	Ferret$_{\mathrm{M}}$	Ferret$_{\mathrm{M+}}$
Vanilla	*oacc*	81.59$_{\pm 0.12}$	21.69$_{\pm 1.01}$	51.25$_{\pm 0.54}$	51.59$_{\pm 0.66}$	48.81$_{\pm 0.74}$	42.05$_{\pm 0.97}$	63.68$_{\pm 0.56}$	**80.46**$_{\pm 0.22}$
	tacc	15.68$_{\pm 0.72}$	10.24$_{\pm 0.82}$	14.35$_{\pm 1.08}$	14.11$_{\pm 0.42}$	15.28$_{\pm 0.39}$	12.02$_{\pm 0.36}$	15.07$_{\pm 0.68}$	**15.83**$_{\pm 0.9}$
ER [12]	*oacc*	79.84$_{\pm 0.09}$	24.51$_{\pm 0.69}$	42.59$_{\pm 0.41}$	43.2$_{\pm 0.41}$	-	40.9$_{\pm 0.83}$	61.57$_{\pm 0.47}$	**78.68**$_{\pm 0.11}$
	tacc	24.54$_{\pm 0.49}$	14.91$_{\pm 0.52}$	19.4$_{\pm 0.68}$	20.32$_{\pm 0.14}$	-	16.8$_{\pm 0.72}$	22.06$_{\pm 0.11}$	**24.32**$_{\pm 0.81}$
MIR [3]	*oacc*	79.84$_{\pm 0.09}$	24.51$_{\pm 0.69}$	42.53$_{\pm 0.49}$	43.1$_{\pm 0.35}$	-	40.9$_{\pm 0.83}$	61.57$_{\pm 0.47}$	**78.68**$_{\pm 0.11}$
	tacc	24.54$_{\pm 0.49}$	14.91$_{\pm 0.52}$	19.78$_{\pm 0.66}$	20$_{\pm 0.51}$	-	16.8$_{\pm 0.72}$	22.06$_{\pm 0.11}$	**24.32**$_{\pm 0.81}$
LWF [47]	*oacc*	81.6$_{\pm 0.13}$	21.7$_{\pm 1.02}$	51.25$_{\pm 0.54}$	51.61$_{\pm 0.67}$	-	42.11$_{\pm 0.85}$	63.55$_{\pm 0.49}$	**80.56**$_{\pm 0.28}$
	tacc	15.68$_{\pm 0.72}$	10.24$_{\pm 0.82}$	14.35$_{\pm 1.08}$	14.11$_{\pm 0.42}$	-	12$_{\pm 0.36}$	15.07$_{\pm 0.68}$	**15.83**$_{\pm 0.9}$
MAS [2]	*oacc*	81.66$_{\pm 0.18}$	22.11$_{\pm 0.7}$	51.19$_{\pm 0.42}$	51.62$_{\pm 0.75}$	-	41.8$_{\pm 0.86}$	63.57$_{\pm 0.59}$	**80.48**$_{\pm 0.21}$
	tacc	16.76$_{\pm 0.53}$	10.54$_{\pm 0.59}$	13.75$_{\pm 0.68}$	13.98$_{\pm 1.11}$	-	11.56$_{\pm 0.09}$	14.87$_{\pm 0.73}$	**16.48**$_{\pm 0.98}$



(a) CORe50/ConvNet (b) CLEAR100/ResNet

Figure 10. Consumed memory of different stream learning algorithms. Ferret achieves rapid adaptation across varying memory constraints.



(a) CLEAR100/ResNet (b) CLEAR100/MobileNet

Figure 11. Relationships between online accuracy and memory consumption of different pipeline parallelism strategies, the marker size represents the standard errors of means.

sumption may not hold in scenarios where the data distribution changes significantly over a short time (*e.g.*, emergency handling). In such cases, λ in Ferret requires manual tuning to ensure optimal performance.