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

Yuhao Zhou[1] Yuxin Tian[1] Jindi Lv[1] Mingjia Shi[2] Yuanxi Li[3]
Qing Ye[1†] Shuhao Zhang[4] Jiancheng Lv[1]
[1]Sichuan University [2]National University of Singapore
[3]University of Illinois Urbana-Champaign [4]Huazhong University of Science and Technology

Abstract

In the realm of high-frequency data streams, achieving real-time learning within varying memory constraints is paramount. This paper presents Ferret, a comprehensive framework designed to enhance online accuracy *of Online Continual Learning (OCL) algorithms while dynamically adapting to varying memory budgets. Ferret employs a fine-grained pipeline parallelism strategy combined with an iterative gradient compensation algorithm, ensuring seamless handling of high-frequency data with minimal latency, and effectively counteracting the challenge of stale gradients in parallel training. To adapt to varying memory budgets, its automated model partitioning and pipeline planning optimizes performance regardless of memory limitations. Extensive experiments across 20 benchmarks and 5 integrated OCL algorithms show Ferret's remarkable efficiency, achieving up to 3.7× lower memory overhead to reach the same online accuracy compared to competing methods. Furthermore, Ferret consistently outperforms these methods across diverse memory budgets, underscoring its superior adaptability. These findings position Ferret as a premier solution for efficient and adaptive OCL framework in real-time environments.*

1. Introduction

Data is crucial for Machine Learning (ML), forming the basis for algorithms and models [30, 53, 75]. In real-world applications [23, 71], data arrives in high-frequency streams with varying distributions [52, 78]. This makes data time-sensitive and short-lived [11, 48, 76], rendering offline-trained models based on historical data ineffective for future data of unknown distribution [61]. Thus, the significance of *Online Continual Learning* (OCL) is growing [6, 40, 77], as it enables learning over data streams to adapt to dynamic data distributions in real-time.

In the literature, OCL tackles two main challenges: 1) mitigating catastrophic forgetting [51], where the model retains previously learned knowledge while acquiring new information (*e.g.*, regularization-based [2, 9, 10, 19], replay-based [12, 39, 68], sampling-based [3, 4, 83], others [25, 65], etc.), and 2) enhancing rapid adaptation [11, 48], which involves swiftly adjusting to new data or tasks (*e.g.*, latency-oriented [29, 70], buffering [54, 82], others [27, 69], etc.). In general, the increasing demand for resource-limited systems that can seamlessly integrate new information with minimal latency has driven the popularity of OCL [32] Therefore, this paper explores the challenge of rapid adaptation under varying memory constraints in OCL.

To effectively address the above OCL challenge, it is essential to explore solutions beyond mentioned algorithmic improvements by also optimizing the underlying framework. An efficient OCL framework must prioritize both processing speed and memory management under the limited memory capacity so that it can efficiently handle unlimited data streams with dynamic data distributions for increased *online accuracy* [11] (*i.e.*, a metric measuring real-time accuracy for continuous new data predictions). Specifically, the framework should quickly process incoming data to extract valuable insights and make informed decisions [56, 74] by minimizing both the latency from data receipt to its initial processing and the time taken for the learning process itself. Additionally, the framework is not only expected to operate within a predetermined memory allotment but also to demonstrate scalability across diverse memory capacities [18, 57]. This duality ensures that the framework remains efficient regardless of the memory resources available, thereby maintaining consistent performance in dynamic environments.

Numerous ML frameworks have been proposed that offer innovative approaches to scalable and flexible ML development [5, 17, 31, 36, 37, 42, 55, 84, 86]. For instance, Ray [55] facilitates distributed computing on any scale, while Pytorch [5] excels in dynamic computation graphs for model training. Despite their advancements, these frameworks often do not specifically address the unique requirements of learning over streaming data [29], which is a key focus of OCL. Recently, there are some frameworks dedicated to OCL by prioritizing real-time data processing [44, 46, 80]. Nevertheless, they either lack general applicability or fail to balance processing speed with consumed memory, leading to reduced online accuracy and low memory scalability, underscoring the need for innovative solutions in this domain.

In this work, we propose an OCL framework named Ferret, designed to achieve e**F**fici**E**nt pipeline lea**R**ning over f**R**equ**E**nt data s**T**reams for enhanced online accuracy across memory constraints. Ferret comprises a fine-grained pipeline parallelism component with an iterative gradient compensation algorithm and a model partitioning and pipeline planning component. Firstly, to facilitate rapid adaptation over frequent streaming data for higher online accuracy, Ferret employs a fine-grained pipeline parallel strategy, allowing precise control over each pipeline stage for seamless data management. Additionally, to mitigate the impact of stale gradients in parallel processing, Ferret integrates a novel iterative gradient compensation algorithm. Secondly, to guide the selection of optimal model partition schemes and pipeline configurations under given memory budgets, Ferret solves the involved multivariate optimization problem through a bi-level optimization algorithm.

Our contributions can be outlined as follows:

- We propose a framework named Ferret for boosting the online accuracy of OCL algorithms under memory constraints. To the best of our knowledge, this is the first work focusing on enhancing OCL by employing pipeline parallelism and scheduling.
- To process high-frequent data streams without delay, Ferret employs a fine-grained pipeline parallelism strategy, enabling interleaved processing of incoming streaming data. Furthermore, Ferret utilizes an iterative gradient compensation algorithm to efficiently mitigate the effects of stale gradients across different pipeline stages, preventing performance degradation.
- We derive the optimal parameters for automatic model partitioning and pipeline planning by mapping the involved multi-variable optimization problem into a bi-level optimization problem.
- Extensive experiments on 20 benchmarks demonstrate that our proposed framework consistently enables more efficient OCL within given memory budgets. The code is open-sourced for reproduction.

2. Related Work

The current OCL research focuses on two areas: mitigating catastrophic forgetting and enhancing rapid adaptation.

Mitigating catastrophic forgetting: Catastrophic forgetting, often quantified by the test accuracy [29, 46, 48], poses a significant barrier to the efficacy of OCL in dynamic environments, where the ability to preserve historical information is crucial. Multiple directions have emerged to reduce catastrophic forgetting, including: 1) regularization-based techniques [2, 9, 10, 19] impose constraints on weight updates to preserve important parameters that are crucial for past tasks. 2) replay-based techniques [12, 39, 68] help the model to rehearse old knowledge alongside new information by maintaining a memory of previous data. 3) sampling-based techniques [3, 4, 83] enhance the efficiency of replay mechanisms by selectively choosing the most relevant data samples for rehearsal. 4) other techniques [25, 65] focus on various novel approaches, such as modular networks and dynamically allocated resources, to protect previously learned information from being overwritten.

Enhancing rapid adaptation: Rapid adaptation is in scenarios where immediate processing of incoming data is required [11, 48], which is often quantified by the online accuracy [11] defined as $oacc_{\mathcal{A}}(t) = \sum_{i=1}^{t} acc(y^i, \hat{y}^i)/t$. Strategies developed to enhance rapid adaptation include: 1) latency-oriented techniques [29, 70] iteratively generate predictions and update model parameters immediately upon the arrival of streaming data by discarding data that cannot be processed in time. 2) buffering-oriented techniques [54, 82] buffers and samples incoming data streams and apply periodic batch-training [62]. 3) other techniques [27, 69] introduce novel methods like adapting model structures in response to new tasks and learning how to learn efficiently, to adapt to dynamic data distributions rapidly.

3. Motivation

To effectively navigate the challenges posed by OCL, it is crucial to expand our approach beyond merely refining mentioned OCL algorithms, by also enhancing the underlying ML framework to adaptively balance processing speed with efficient memory management under memory constraints. Particularly, boosting processing speed is essentially reducing data processing time, which can be represented as $t^l + F/R_h P_h$, where t^l denotes the latency from data arrival to processing, F denotes the required floating point operations (FLOPS) by the underlying OCL algorithm, R_h and P_h denote the hardware utilization rate and the theoretical floating point operations per second (FLOPs) of the hardware, respectively. Clearly, only t^l and R_h are optimizable by the framework.

Existing ML frameworks mainly focus on: 1) distributed and parallel computing [42, 55, 86], 2) Optimized model training and deployment [5, 36, 84], and 3) others including security [17, 37] and debugging [31]. These frameworks facilitate scalable and flexible ML, yet they rarely tackle the challenges of managing streaming data. Regrettably, the few ML frameworks designed for OCL [44, 46, 80] either lack general applicability or fail to concurrently optimize t^l and R_h within memory limitations. For instance, Kraken [80] is tailored for recommendation systems. Conversely, while Camel [46] and LifeLearner [44] boost R_h via buffering and sampling, they also raise t^l and memory usage, reducing online accuracy.

Pipeline parallelism can naturally process sequential streaming data while utilizing batch training. This motivates us to incorporate pipeline parallelism into OCL to simultaneously minimize t^l and maximize R_h under a given memory budget, thereby boosting online accuracy. We achieve this balance through refined scheduling strategies and better hardware integration, ensuring optimal resource utilization within the constraints of memory budgets.

4. Problem Formulation

In this section, we define the problem we aim to address. Note that the notations used throughout this paper are defined in Sec. 9 in the appendix.

Consider a general learning problem defined over a feature space $\mathcal{X}$ and a label space $\mathcal{Y}$ that aims to minimize a loss function $\mathcal{L}(D^t;\theta)$ where data $D^t = (\mathbf{x}^t, \mathbf{y}^t) \in \mathcal{X} \times \mathcal{Y}$ arrives at timestamp t. Our objective is to rapidly derive an updated model θ^t with D^t and θ^{t-1} under a given memory constraint M, so that the online accuracy of θ^t is high. Unlike updating a model offline with a pre-collected dataset, D^t will be discarded after updating θ^{t-1} in OCL.

Directly optimizing online accuracy in our objective during runtime is hard, as the online accuracy is only calculable after obtaining labels of incoming data. Instead, we measure the volume of data values learned by the model as a proxy to estimate and optimize online accuracy. Formally, assuming D^t has an initial data value of V_{D^t} and its data value declines as a time-dependent exponential decay function [76], we define the Adaptation Rate as follows.

Definition 4.1 (Adaptation Rate of A OCL framework)**.** Consider a OCL framework $\mathcal{A}$ receives a data $D^t = (\mathbf{x}^t, \mathbf{y}^t)$ at timestamp t that has an initial data value of V_{D^t}, and updates a model θ^{t-1} in the hypothesis space Θ at timestamp $t + r^t_{\mathcal{A}}$ ($r^t_{\mathcal{A}} = +\infty$ if D^t is discarded). Let the data value of D^t decline as a time-dependent exponential decay function, and new data D^t constantly arrives until $t = T$. The Adaptation Rate of $\mathcal{A}$ is defined as

$$R^T_{\mathcal{A}} = \frac{\sum_{t=0}^{T} e^{-cr^t_{\mathcal{A}}} V_{D^t}}{T}, \tag{1}$$

where the constant c describes the reduction rate of V_{D^t}.

With Def. 4.1, our objective can be formulated as

$$\max_{\mathcal{A}} \mathcal{R}^T_{\mathcal{A}} \text{ s.t. } \mathcal{M}_{\mathcal{A}} \le M, \tag{2}$$

where $\mathcal{M}_{\mathcal{A}}$ is the memory footprint of $\mathcal{A}$ during training.

5. Methodology

The workflow of Ferret is shown in Fig. 1, comprising a fine-grained pipeline parallelism component (A), followed by a model partitioning and pipeline planning component (B). In A, the model is trained using Ferret's fine-grained pipeline parallelism to manage high-frequent data streams with minimal latency. Given the high degree of parallelism within the system, gradient staleness can become significant and variable, potentially causing severe model degradation. To mitigate this issue, an iterative gradient compensation algorithm is applied prior to model updating. In B, the model is profiled to optimize Eq. 2, determining the optimal model partition scheme and pipeline configuration.

5.1. Fine-grained Pipeline Parallelism

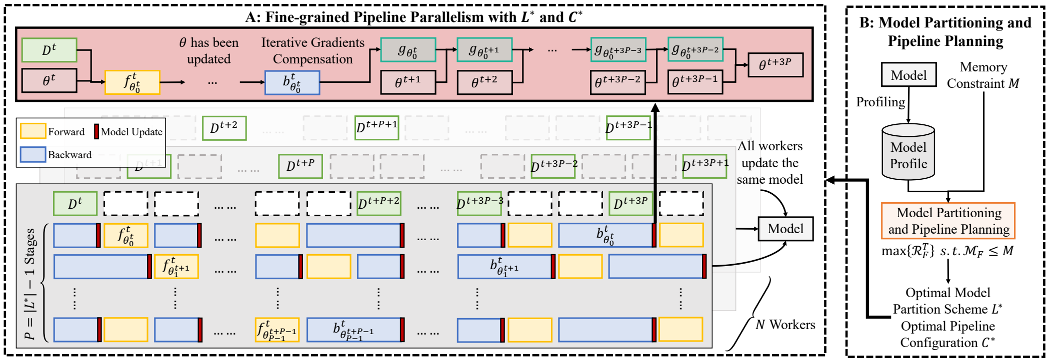
5.1.1 Architectural design

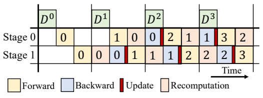
Ferret utilizes an asynchronous pipeline parallelism strategy with 1F1B scheduling to process streaming data immediately upon arrival. To efficiently handle high-frequency data streams without delay, it is imperative that $t^f + t^b$ is minimized. However, $t^f + t^b$ is inherently lower bounded by $(\max_i \hat{t}^f_i + \max_i \hat{t}^b_i)$, indicating that some of the data must be discarded if t^d is less than this lower bound. To prevent the loss of data, Ferret enhances system throughput by deploying $N \le \lceil (t^f + t^b)/t^d \rceil$ workers, each performing pipeline parallelism concurrently over interleaved data streams. Specifically, the i-th data is processed by the n-th worker if and only if $i \equiv c^d_n \mod \lceil (t^f + t^b)/t^d \rceil$. This strategy, while effective in reducing latency, significantly increases memory usage. Therefore, Ferret balances the trade-offs between $\mathcal{R}$ and $\mathcal{M}$ by collectively employing four **t**echniques: activation recomputation [13], gradient accumulation [59], back-propagation omission and worker removal, allowing precise control over each pipeline stage for seamless data management.

T1. Activation Recomputation: Activation recomputation exchanges additional computational overhead for reduced memory usage, as Fig. 1a illustrated. In Ferret, a binary indicator c^r_n within configuration C denotes whether activation recomputation is enabled for the n-th worker. When activation recomputation is enabled (*i.e.*, $c^r_n = 1$), an additional forward pass is executed prior to the backward pass, effectively managing memory consumption at the expense of increased computational load.

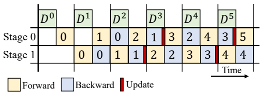
T2. Gradient Accumulation: Gradient Accumulation allows multiple forward and backward passes to accumulate gradients before model updating, thereby decreasing the frequency of parameter updates, as Fig. 1b depicted. In Ferret, the parameter $c^a_{n,j}$ in configuration C defaults to 1, indicating the number of gradient accumulation steps before model updating for the j-th stage in the n-th worker. By utilizing gradient accumulation, the j-th stage in the n-th worker only stores $(1 + \lceil (P - j - 1)/c^a_{n,j} \rceil)$, instead of $(P - j)$, models, thereby optimizing memory usage.

T3. Back-propagation Omission: To further reduce memory usage, back-propagation omission skips all back-

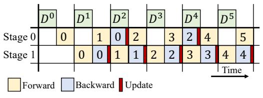




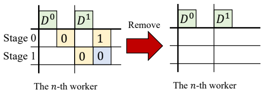
(a) Activation Recomputation (T1)



(b) Gradient Accumulation (T2)



(c) Backpropagation Omission (T3)



(d) Worker Removal (T4)

Figure 1. The overall workflow of Ferret. In A, based on the optimal model partition scheme L^* and pipeline configuration C^*, N workers are spawned to initiate fine-grained pipeline parallelism that consumes streaming data interleavedly, and update the same model asynchronously by iteratively compensating stale gradients. In B, L^* and C^* are obtained by optimizing Eq. 2.

ward passes that depend on previous model parameters, as Fig. 1c illustrated. In Ferret, the parameter $c_{n,j}^o$ in configuration C defaults to 0, indicating the number of backpropagation omission steps for the j-th stage in the n-the worker. This approach reduces memory overhead by eliminating the need to store multiple versions of models.

T4. Worker Removal: Spawning N workers increases the system throughput but also linearly increases the memory footprint. When resources are highly constrained, the n-th worker can be shut down and removed to reduce the memory overhead by setting $c_n^d = -1$ in configuration C.

Finally, assume the initial data value of any data is V_D, and the retained value of the data when updating a subset of model parameters is proportional to the size of the subset model parameters, given L and C, $\mathcal{R}$ and $\mathcal{M}$ of the fine-grained pipeline parallelism strategy can be respectively formulated as

$$\mathcal{R}_F^T = \sum_{\substack{n=1\\ c_n^d \geq 0}}^{N-1} \sum_{i=0}^{P-1} \frac{|w_i|}{\sum_{j=0}^{P-1}(|w_j|)} \frac{1}{c_{n,i}^a} \sum_{j=0}^{c_{n,i}^a-1} A_{i,j}, \text{where } A_{i,j} = \frac{e^{-c((P+j)t^f+(P-i+j)t^b+c_n^r(P-i+j)t^f)}V_D}{LCM(\{c_{n,k}^o+1|k\in[i,P-1]\})(t^f+t^b+c_n^r t^f)}, \quad (3)$$

$$\mathcal{M}_F = \sum_{\substack{n=1\\ c_n^d \geq 0}}^{N-1} \sum_{i=0}^{P-1} (1+\lceil \frac{P-i-1}{c_{n,i}^a} \rceil - c_{n,i}^o)(|w_i| + |a_i| - c_n^r \sum_{l=L_i+1}^{L_{i+1}-1} |\hat{a}_l|), \quad (4)$$

where $LCM(\cdot)$ denotes the Least Common Multiple.

5.1.2 Iterative Gradient Compensation

Since fine-grained pipeline parallelism is asynchronous, the model will be inevitably updated by stale gradients, leading to performance degradation. Moreover, different pipeline stages of the model are updated by gradients with varying staleness. To surmount the above challenges, Ferret firstly proposes to efficiently approximate $\nabla\mathcal{L}(D^{t-1};\theta^t)$ using $\nabla\mathcal{L}(D^{t-1};\theta^{t-1})$ by a cost-effective approximator $A_{\mathcal{I}}(\cdot)$ based on Taylor series expansion and the Fisher information matrix. Then, we extend this approximator to approximate $\nabla\mathcal{L}(D^{t-1};\theta^{t+\tau-1})$ using $\nabla\mathcal{L}(D^{t-1};\theta^{t-1})$ by iteratively applying $A_{\mathcal{I}}(\cdot)$.

Gradients Compensation via Taylor Series Expansion: In prior work, $\nabla\mathcal{L}(D^{t-1};\theta^t)$ was naively set to $\nabla\mathcal{L}(D^{t-1};\theta^{t-1})$ [58, 59] and can be regarded as a zero-order Taylor series expansion, leading to a high approximation error $||\nabla\mathcal{L}(D^{t-1};\theta^t) - \nabla\mathcal{L}(D^{t-1};\theta^{t-1})||^2$. To reduce the approximation error, we expand $\nabla\mathcal{L}(D^{t-1};\theta^t)$ at θ^{t-1} by a first-order Taylor series expansion as follows:

$$\nabla\mathcal{L}(D^{t-1};\theta^t) \approx \nabla\mathcal{L}(D^{t-1};\theta^{t-1}) + \mathbb{H}(\mathcal{L}(D^{t-1};\theta^{t-1})) \odot (\theta^t - \theta^{t-1}), \quad (5)$$

where $\mathbb{H}(\cdot)$ denotes the Hessian matrix of $\cdot$. Previous works have revealed that the Fisher information matrix (FIM) serves as an approximation of the Hessian matrix if $\mathcal{L}(\cdot)$ is a negative log-likelihood loss [28, 63]. Assuming θ^t gradually converges to its optimal value θ^* during training, we can achieve an unbiased estimation of $\mathbb{H}(\cdot)$ by:

$$\epsilon_t \triangleq \mathbb{E}_{D,\theta^*} ||\mathcal{I}(\theta^t) - \mathbb{H}(\mathcal{L}(\cdot;\theta^t))|| \to 0, t \to +\infty, \quad (6)$$

where $\mathcal{I}(\theta)$ is the FIM. To further mitigate space complexity, $\mathcal{I}(\theta)$ is approximated by its diagonal elements with a hyper-parameter λ to control variance, *i.e.*,

$$\mathbb{H}(\mathcal{L}(\cdot;\theta^t)) \approx \lambda \nabla\mathcal{L}(\cdot;\theta^{t-1}) \odot \nabla\mathcal{L}(\cdot;\theta^{t-1})^\top. \quad (7)$$

Incorporating Eq.7 into Eq.5, we obtain:

$$\nabla\mathcal{L}(D^{t-1};\theta^t) \approx A_{\mathcal{I}}(\nabla\mathcal{L}(D^{t-1};\theta^{t-1}), \theta^t, \theta^{t-1}) = \quad (8)$$
$$\nabla\mathcal{L}(D^{t-1};\theta^{t-1}) + \lambda \nabla\mathcal{L}(D^{t-1};\theta^{t-1}) \odot \nabla\mathcal{L}(D^{t-1};\theta^{t-1})^\top \odot \Delta\theta,$$

where $A_{\mathcal{I}}(\cdot)$ serves as the approximator to compensate $\cdot$.

Iterative Compensation: More generally, to approx-

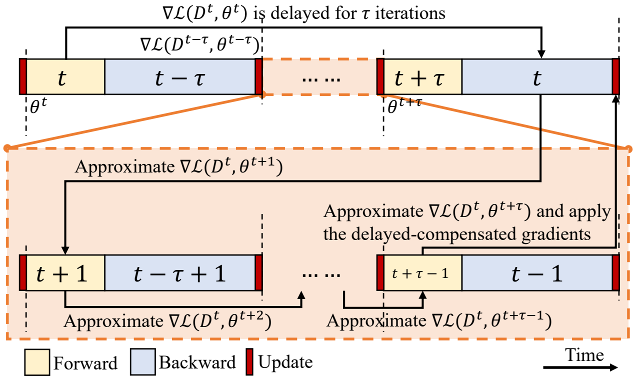


Figure 2. To adapt to different levels of staleness in fine-grained pipeline parallelism, $\nabla\mathcal{L}(D^t,\theta^{t+\tau})$ is iteratively approximated by $\nabla\mathcal{L}(D^t,\theta^t)$.

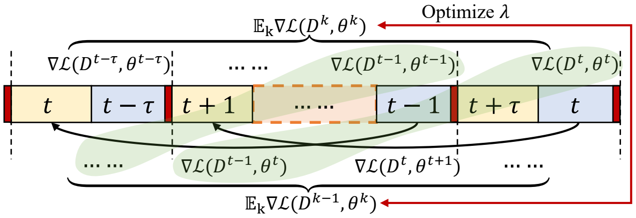


Figure 3. To further reduce approximation errors, we optimize λ automatically by comparing historical approximations ($\nabla\mathcal{L}(D^t,\theta^t)$, etc.) and observations ($\nabla\mathcal{L}(D^{t-1},\theta^t)$, etc.)

imate $\nabla\mathcal{L}(D^{t-1};\theta^{t+\tau-1})$ using $\nabla\mathcal{L}(D^{t-1};\theta^{t-1})$, Ferret proposes an iterative application of $A_{\mathcal{I}}(\cdot)$, as depicted in Fig. 2. This iterative process is defined as follows:

$$\begin{aligned}&\nabla\mathcal{L}(D^{t-1};\theta^{t+\tau-1}) \approx A_{\mathcal{I}}(\nabla\mathcal{L}(D^{t-1};\theta^{t+\tau-2}),\theta^{t+\tau-1},\theta^{t+\tau-2})\\ &\approx A_{\mathcal{I}}(\dots A_{\mathcal{I}}(\nabla\mathcal{L}(D^{t-1};\theta^{t-1}),\theta^t,\theta^{t-1})\dots,\theta^{t+\tau-1},\theta^{t+\tau-2}).\end{aligned} \tag{9}$$

However, this iterative process introduces a cascade of errors, wherein the approximation error $||g^t - A_{\mathcal{I}}(g^{t-1},\theta^t,\theta^{t-1})||^2$ is propagated and amplified with each successive approximation. This arises because each approximation depends on the output of the preceding one.

To mitigate this problem, we propose to optimize λ under the mild assumption that the distributions of $\mathbb{E}_k D^k$ and $\mathbb{E}_k D^{k+1}$ are similar, as illustrated in Fig. 3. Thus, the objective of minimizing the approximation error of iterative gradients compensation can be formulated as follows:

$$\begin{aligned}&\min_{\lambda}\mathbb{E}_k||\nabla\mathcal{L}(D^k;\theta^k) - A_{\mathcal{I}}(\nabla\mathcal{L}(D^{k-1},\theta^k,\theta^{k-1}))||^2 + \nu||\lambda||^2\\ &= \min_{\lambda}||D-E-\lambda F||^2 + \nu||\lambda||^2,\end{aligned} \tag{10}$$

where $D = \mathbb{E}_k\nabla\mathcal{L}(D^k;\theta^k)$, $E = \mathbb{E}_k\nabla\mathcal{L}(D^{k-1};\theta^{k-1})$,
$F = \mathbb{E}_k\nabla\mathcal{L}(D^{k-1};\theta^{k-1}) \odot \nabla\mathcal{L}(D^{k-1};\theta^{k-1})^\top \odot (\theta^k - \theta^{k-1})$,

where $\nu||\lambda||^2$ is an ℓ_2 regularization term to constrain the solution of λ for better stability. To reduce memory overhead, D and E can be approximated by Exponential Moving Average (EMA), *i.e.*,

$$\mathbb{E}_k\nabla\mathcal{L}(D^k;\theta^k) = \alpha\mathbb{E}_k\nabla\mathcal{L}(D^{k-1};\theta^{k-1}) + (1-\alpha)\nabla\mathcal{L}(D^{k-1};\theta^{k-1}), \tag{11}$$

where α is the EMA coefficient. Hence, we have

$$D - E = (1-\alpha)(\nabla\mathcal{L}(D^{k-1};\theta^{k-1}) - \mathbb{E}_k\nabla\mathcal{L}(D^{k-1};\theta^{k-1})). \tag{12}$$

Convergence: Similar to the analyses in [85], our iterative gradient compensation algorithm yields convergence rates of $\mathcal{O}(V_1^2\tau/T)$ and $\mathcal{O}(V_2/\sqrt{T})$ for convex and non-convex case, respectively. Here, V_1 and V_2 represent the upper-bound of the $||\cdot||^2$ norm and the variance of the delay-compensated gradient $A_{\mathcal{I}}(\cdot)$, accordingly. Compared to the work in [85], Ferret fixes τ to 1, and minimizes V_1 and V_2 by Eq. 10, boosting algorithm's robustness and accelerating the convergence of the model.

Algorithm Design: The algorithm of Ferret's iterative gradient compensation is illustrated in Alg. 1 in the appendix. Since the maximum possible τ equals $(P-1)$, the time complexity of the algorithm is $\mathcal{O}(P-1)$, which is considered negligible during model training. Moreover, since two additional variables, v_r and v_a, are stored in memory for optimizing λ, the space complexity of this algorithm is $\mathcal{O}(2\sum_{j=0}^{P-1}|w_i|)$. However, by setting $\eta_\lambda = 0$, the optimization of λ is effectively terminated, and λ remains fixed at λ^0. This adjustment allows for manual tuning of λ and eliminates the need for v_r and v_a, thereby increasing flexibility and avoiding additional memory overhead.

5.2. Model Partitioning and Pipeline Planning

The objective of model partitioning and pipeline planning is to find an optimal model partition scheme L^* and its corresponding pipeline configuration C^* that maximize $\mathcal{R}$ within a given memory constraint M, namely,

$$L^*, C^* = \arg\max_{L,C}\mathcal{R}_F^T \ \text{ s.t. } \ \mathcal{M}_F \leq M. \tag{13}$$

This problem can be reformulated as a bi-level optimization problem, decomposing it into two interrelated subproblems: (1) determining the optimal C given a L, and (2) identifying the optimal L based on the solution from (1):

$$\begin{aligned}&L^* = \arg\max_{L}\{\mathcal{R}_F^T|C_L^*\}\\ &s.t.\ \ C_L^* = \arg\max_{C}\{\mathcal{R}_F^T|L\}, \mathcal{M}_F \leq M.\end{aligned} \tag{14}$$

5.2.1 Iterative Configuration Search (Sub-problem 1)

Given a model partition scheme, the objective of subproblem (1) is to solve

$$C^* = \arg\max_{C}\{\mathcal{R}_F^T|L\} \ \text{ s.t. } \ \mathcal{M}_F \leq M. \tag{15}$$

With more than $2^{N(P+1)}$ potential combinations for C, a brute-force enumeration of C is impractical. Observing that $\mathrm{d}\mathcal{M}_F/\mathrm{d}\max_C\{\mathcal{R}_F^T|L\} \geq 0$, we employ an iterative algorithm to determine the optimal C that maximize $\mathcal{R}_F^T$ while ensuring $\mathcal{M}_F$ remains within the memory budget. Specifically, to prevent memory over-consumption, we progressively deploy **T1-T4** as follows to balance $\mathcal{R}_F^T$ 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 Eq. 19 in the appendix.

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 $\Delta_{c_{n,j}^a \rightarrow c_{n,j}^a+1} \mathcal{M}_F = 0$ due to the ceiling function. Consequently, $\mathcal{R}_F^T$ and $\mathcal{M}_F$ will be respectively decreased by Eq. 20 in the appendix.

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 Eq. 21 in the appendix.

S4. Deploy T4 for the n-th worker: If $c_{n,j}^o \neq 0$ for all $j \in [0, p-1)$, removing the n-th worker will lead to a decrease in $\mathcal{R}_F^T$ and $\mathcal{M}_F$ by Eq. 22 in the appendix.

Algorithm Design: The algorithm of the proposed searching is illustrated in Alg. 2 in the appendix. Overall, the time complexity of this algorithm is $\mathcal{O}(NP^2)$, and it will be executed only once before fine-grained pipeline parallelism begins.

5.2.2 Brute-force Planning (Sub-problem 2)

In Ferret, L is determined by first establishing an upper bound on the time consumed for each stage (t^c), and then solving the following optimization problem:

$$L = \arg\min_{L}\{P\} \;\; s.t. \;\; t^f + t^b \leq t^c. \tag{16}$$

Namely, minimizing the number of pipeline stages while ensuring the time consumed for each stage is bounded. Since the layers in a stage must be consecutive, this problem can be solved in linear time by iteratively grouping consecutive layers into a stage until no additional adjacent layer can be grouped. Therefore, the solution space for L is not extensive, being limited to $(\hat{L}^2 - \hat{L})/2$ at worst. Thus, to solve sub-problem (2), we can simply enumerate all possible model partition schemes, feeding them into Alg. 2 in the appendix to obtain the global optimum L^*.

Algorithm Design: The algorithm of the proposed planning is illustrated in Alg. 3 in the appendix. The time complexity of this algorithm is $\mathcal{O}(\hat{L}^3)$. Nevertheless, the algorithm will be executed only once before fine-grained pipeline parallelism begins.

6. Experiments

In this section, we seek answers to the following questions. (1) How does Ferret boost online accuracy? (Sec. 6.2) (2) How does Ferret mitigate catastrophic forgetting. (Sec. 6.2) (3) How does our fine-grained pipeline parallelism perform? (Sec. 6.3) (4) What are influences of different pipeline configurations? (Sec. 6.3) (5) How does our iterative gradient compensation algorithm perform?(Sec. 6.4)

6.1. Evaluation Setup

Datasets and models: Following the conventions of the community [46], 18 image classification datasets, including MNIST [22], FMNIST [79], CIFAR10 [43], CIFAR100 [43], SVHN [60], Tiny-ImageNet [45], CORe50 [50], CORe50-iid, Split-MNIST, Split-FMNIST, Split-CIFAR10, Split-CIFAR100, Split-SVHN, Split-Tiny-ImageNet, Covertype [8], CLEAR10 [48], CLEAR100 [48], are used in our experiments. More details about the datasets can be found in the appendix. To cover both simple and complicated learning problems, five models including Multi-Layer Perceptron (MLP), MNISTNet, ConvNet, ResNet-18 [34] and MobileNet [35] are used in the experiments. Note that ResNet-18 and MobileNet are pretrained on the ImageNet-1K dataset [21].

Compared methods: For question (1): **Oracle**, **1-Skip** [29], **Random-N B-Skip**, **Last-N B-Skip** and Camel [46]. Here, Oracle is an ideal method that sequentially processes every streaming data without any delay. B-Skip and Camel selects a subset from the latest B unprocessed data using Random-N, Last-N and Coreset sampler, respectively. For question (2): **Vanilla**, **ER** [12], **MIR** [3], **LwF** [47], **MAS** [2]. For question (3) and (4): **DAPPLE** [24], **Pipedream** [58], **Pipedream$_{2BW}$** [59], **Zero-Bubble** [66] and **Hanayo** [49]. For question (5): **None**, **Step-Aware** [33, 41], **Gap-Aware** [7], **Fisher** [14], and **Iter-Fisher** (*i.e.*, iterative gradient compensation).

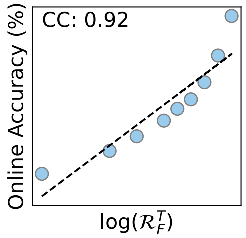


Figure 7. Relation between $oacc$ and $\log(\mathcal{R}_F^T)$

Evaluation metrics: To measure catastrophic forgetting while accounting for memory footprint, Test Accuracy [29, 46] Gain per unit of Memory[1] (the higher the better) is defined as

$$tagm_{\mathcal{B}}(\mathcal{A}, t) = \log(\frac{\exp(tacc_{\mathcal{A}}(t) - tacc_{\mathcal{B}}(t))}{\mathcal{M}_{\mathcal{A}}/\mathcal{M}_{\mathcal{B}}}), \tag{17}$$

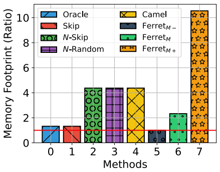
where $tacc$ computes the test accuracy and $\mathcal{B}$ is the baseline

---

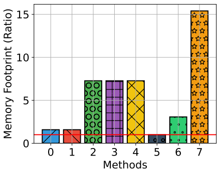
[1] Results in the appendix show standard test accuracy and online accuracy but ignore memory consumption during training.

Table 1. Online Accuracy Gain per unit of Memory ($agm_{\mathcal{B}}(\mathcal{A}, T)$) of different algorithms, where $\mathcal{B}$ is the 1-Skip. "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	$27.32_{\pm0.71}$	$0_{\pm0}$	$-0.43_{\pm0.6}$	$-0.26_{\pm0.14}$	$-0.71_{\pm0.32}$	$5.31_{\pm0.7}$	$\underline{16.26}_{\pm0.37}$	$\mathbf{26.34}_{\pm0.7}$
FMNIST/MNISTNet	$19.35_{\pm0.99}$	$0_{\pm0}$	$-0.31_{\pm0.47}$	$-0.25_{\pm0.5}$	$-0.6_{\pm0.4}$	$5.93_{\pm0.81}$	$\underline{12.69}_{\pm0.81}$	$\mathbf{18.37}_{\pm1.01}$
EMNIST/MNISTNet	$13_{\pm0.48}$	$0_{\pm0}$	$1.94_{\pm0.04}$	$2.02_{\pm0.03}$	$1.55_{\pm0.1}$	$4.19_{\pm0.17}$	$\underline{8.8}_{\pm0.4}$	$\mathbf{12.09}_{\pm0.47}$
CIFAR10/ConvNet	$10.57_{\pm0.09}$	$0_{\pm0}$	$4.71_{\pm0.05}$	$4.78_{\pm0.03}$	$4.7_{\pm0.05}$	$3.21_{\pm0.16}$	$\underline{6.21}_{\pm0.15}$	$\mathbf{9.44}_{\pm0.12}$
CIFAR100/ConvNet	$5.24_{\pm0.01}$	$0_{\pm0}$	$0.78_{\pm0.07}$	$0.83_{\pm0.06}$	$0.75_{\pm0.08}$	$1.58_{\pm0.04}$	$\underline{2.6}_{\pm0.03}$	$\mathbf{4.39}_{\pm0.05}$
SVHN/ConvNet	$15.41_{\pm0.23}$	$0_{\pm0}$	$7.04_{\pm0.08}$	$7.24_{\pm0.11}$	$7.39_{\pm0.09}$	$5_{\pm0.1}$	$\underline{11.52}_{\pm0.23}$	$\mathbf{14.34}_{\pm0.31}$
TinyImagenet/ConvNet	$2.13_{\pm0.07}$	$0_{\pm0}$	$-0.22_{\pm0.04}$	$-0.2_{\pm0.03}$	$-0.21_{\pm0.05}$	$0.48_{\pm0.03}$	$\underline{0.54}_{\pm0.03}$	$\mathbf{1.19}_{\pm0.07}$
CORe50/ConvNet	$26.01_{\pm0.42}$	$0_{\pm0}$	$12.13_{\pm0.42}$	$12.27_{\pm0.44}$	$11.07_{\pm0.48}$	$9.08_{\pm0.45}$	$\underline{17.95}_{\pm0.45}$	$\mathbf{24.49}_{\pm0.43}$
CORe50-iid/ConvNet	$19.24_{\pm2.9}$	$0_{\pm0}$	$2.87_{\pm5.71}$	$5.74_{\pm2.8}$	$5.21_{\pm2.49}$	$3.55_{\pm2.77}$	$\underline{10.74}_{\pm2.76}$	$\mathbf{17.96}_{\pm2.88}$
SplitMNIST/MNISTNet	$18.21_{\pm0.76}$	$0_{\pm0}$	$2.34_{\pm0.43}$	$2.37_{\pm0.63}$	$3.3_{\pm0.48}$	$6.11_{\pm0.84}$	$\underline{14.55}_{\pm0.55}$	$\mathbf{17.05}_{\pm0.72}$
SplitFMNIST/MNISTNet	$11.32_{\pm1.47}$	$0_{\pm0}$	$1.53_{\pm0.47}$	$1.49_{\pm0.42}$	$1.96_{\pm0.39}$	$5.43_{\pm0.56}$	$\underline{9.37}_{\pm1.35}$	$\mathbf{10.29}_{\pm1.47}$
SplitCIFAR10/ConvNet	$7.49_{\pm0.12}$	$0_{\pm0}$	$3.05_{\pm0.11}$	$3.11_{\pm0.11}$	$3.12_{\pm0.09}$	$2.91_{\pm0.19}$	$\underline{4.84}_{\pm0.2}$	$\mathbf{6.19}_{\pm0.07}$
SplitCIFAR100/ConvNet	$10.51_{\pm0.15}$	$0_{\pm0}$	$2.81_{\pm0.07}$	$2.86_{\pm0.05}$	$2.74_{\pm0.13}$	$3.54_{\pm0.03}$	$\underline{6.13}_{\pm0.13}$	$\mathbf{9.61}_{\pm0.04}$
SplitSVHN/ConvNet	$6.49_{\pm0.33}$	$0_{\pm0}$	$2.9_{\pm0.19}$	$2.91_{\pm0.21}$	$2.89_{\pm0.21}$	$2.76_{\pm0.16}$	$\underline{5}_{\pm0.28}$	$\mathbf{5.38}_{\pm0.35}$
SplitTinyImagenet/ConvNet	$2.14_{\pm0.1}$	$0_{\pm0}$	$-0.24_{\pm0.03}$	$-0.21_{\pm0.02}$	$-0.26_{\pm0.01}$	$0.47_{\pm0.01}$	$\underline{0.62}_{\pm0.01}$	$\mathbf{1.19}_{\pm0.06}$
CLEAR10/ResNet	$10.37_{\pm0.06}$	$0_{\pm0}$	$7.84_{\pm0.07}$	$\underline{7.93}_{\pm0.06}$	$-2.9_{\pm10.55}$	$2.44_{\pm0.06}$	$7.71_{\pm0.06}$	$\mathbf{9.26}_{\pm0.08}$
CLEAR10/MobileNet	$20.36_{\pm0.2}$	$0_{\pm0}$	$11.8_{\pm0.22}$	$12_{\pm0.14}$	$11.85_{\pm0.07}$	$-1.77_{\pm0.15}$	$\underline{14.68}_{\pm0.5}$	$\mathbf{18.51}_{\pm0.35}$
CLEAR100/ResNet	$21.71_{\pm0.43}$	$0_{\pm0}$	$15.19_{\pm0.49}$	$15.36_{\pm0.46}$	$14.39_{\pm0.46}$	$7.51_{\pm0.44}$	$\underline{15.53}_{\pm0.35}$	$\mathbf{20.84}_{\pm0.57}$
CLEAR100/MobileNet	$23.51_{\pm1.03}$	$0_{\pm0}$	$9.16_{\pm0.28}$	$9.39_{\pm0.15}$	$8.72_{\pm0.06}$	$1.05_{\pm0.13}$	$\underline{15.8}_{\pm0.39}$	$\mathbf{22.11}_{\pm0.59}$
Covertype/MLP	$7.66_{\pm0.27}$	$0_{\pm0}$	$-1.33_{\pm0.3}$	$-1.3_{\pm0.31}$	$-1.34_{\pm0.29}$	$0.74_{\pm0.21}$	$\underline{1.61}_{\pm0.29}$	$\mathbf{3.38}_{\pm0.42}$

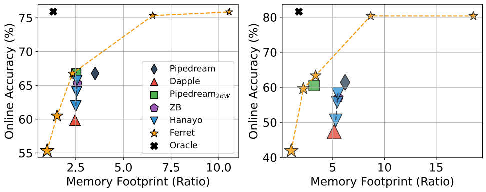


(a) EMNIST/MNISTNet



(b) CIFAR100/ConvNet

Figure 4. Consumed memory of different stream learning algorithms. Ferret achieves rapid adaptation across varying memory constraints.



(a) EMNIST/MNISTNet (b) CORe50/ConvNet

Figure 6. Relationships between online accuracy and memory consumption of different pipeline parallelism strategies, the marker size represents the standard errors of means.

Table 2. Online Accuracy Gain per unit of Memory ($agm_{\mathcal{B}}(\mathcal{A}, T)$) and Test Accuracy Gain per unit of Memory ($tagm_{\mathcal{B}}(\mathcal{A})$) of different integrated OCL algorithms on CORe50/ConvNet, where $\mathcal{B}$ is the 1-Skip. 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	agm	$26.01_{\pm0.42}$	$0_{\pm0}$	$12.13_{\pm0.42}$	$12.27_{\pm0.44}$	$11.07_{\pm0.48}$	$9.08_{\pm0.45}$	$\underline{17.21}_{\pm0.45}$	$\mathbf{24.82}_{\pm0.43}$
	$tagm$	$2.36_{\pm0.64}$	$0_{\pm0}$	$1.08_{\pm0.62}$	$0.97_{\pm0.39}$	$\underline{1.48}_{\pm0.42}$	$1.01_{\pm0.45}$	$1.07_{\pm0.49}$	$\mathbf{1.73}_{\pm0.53}$
ER [12]	agm	$24.03_{\pm0.26}$	$0_{\pm0}$	$7.84_{\pm0.17}$	$8.11_{\pm0.29}$	-	$7.12_{\pm0.09}$	$\underline{16.09}_{\pm0.16}$	$\mathbf{23.5}_{\pm0.25}$
	$tagm$	$4.18_{\pm0.43}$	$0_{\pm0}$	$1.94_{\pm0.26}$	$2.34_{\pm0.29}$	-	$0.82_{\pm0.29}$	$\underline{3.1}_{\pm0.25}$	$\mathbf{4.06}_{\pm0.36}$
MIR [3]	agm	$24.03_{\pm0.26}$	$0_{\pm0}$	$7.82_{\pm0.21}$	$8.06_{\pm0.22}$	-	$7.12_{\pm0.09}$	$\underline{16.09}_{\pm0.15}$	$\mathbf{23.5}_{\pm0.25}$
	$tagm$	$4.18_{\pm0.43}$	$0_{\pm0}$	$2.1_{\pm0.15}$	$2.2_{\pm0.14}$	-	$0.82_{\pm0.29}$	$\underline{3.1}_{\pm0.25}$	$\mathbf{4.06}_{\pm0.36}$
LwF [47]	agm	$26.02_{\pm0.42}$	$0_{\pm0}$	$12.25_{\pm0.42}$	$12.4_{\pm0.45}$	-	$9.02_{\pm0.4}$	$\underline{17.96}_{\pm0.46}$	$\mathbf{24.67}_{\pm0.4}$
	$tagm$	$2.36_{\pm0.64}$	$0_{\pm0}$	$1.2_{\pm0.62}$	$1.09_{\pm0.39}$	-	$0.91_{\pm0.44}$	$\underline{1.88}_{\pm0.49}$	$\mathbf{1.54}_{\pm0.53}$
MAS [2]	agm	$25.86_{\pm0.25}$	$0_{\pm0}$	$12.04_{\pm0.23}$	$12.23_{\pm0.22}$	-	$8.7_{\pm0.17}$	$\underline{17.79}_{\pm0.22}$	$\mathbf{24.46}_{\pm0.23}$
	$tagm$	$2.7_{\pm0.23}$	$0_{\pm0}$	$0.81_{\pm0.41}$	$0.91_{\pm0.24}$	-	$0.59_{\pm0.23}$	$\underline{1.66}_{\pm0.18}$	$\mathbf{1.69}_{\pm0.21}$

method for comparison. Similarly, as Fig. 7 shows online accuracy can be used to estimate $\mathcal{R}_F^T$ [11], Online Accuracy Gain per unit of Memory[1] (the higher the better) is defined as

$$agm_{\mathcal{B}}(\mathcal{A}, t) = \log(\frac{\exp(oacc_{\mathcal{A}}(t) - oacc_{\mathcal{B}}(t))}{\mathcal{M}_{\mathcal{A}}/\mathcal{M}_{\mathcal{B}}}). \tag{18}$$

We evaluate three versions of Ferret under different memory constraints: Ferret$_{\mathrm{M}-}$ (minimal), Ferret$_{\mathrm{M}}$ (the same memory constraint as Pipedream$_{2BW}$), and Ferret$_{\mathrm{M}+}$ (no constraint). Without clarification, each experiment is independently repeated three times to obtain the final results. In all tables, the best and second-best performance are highlighted by **bold** and $\underline{\text{underline}}$, respectively. More details about the evaluation setup can be found in Sec. 12.

6.2. Overall Comparisons

Table 1 shows $agm_{\mathcal{B}}(\mathcal{A}, T)$ across 20 different settings to evaluate both performance and consumed memory of different frameworks. Here, $\mathcal{B}$ is chosen to be the 1-Skip due to its low memory footprint. From the table, it is evident that Ferret$_{\mathrm{M}}$ and Ferret$_{\mathrm{M}+}$ constantly outperform other competing algorithms. Notably, Ferret$_{\mathrm{M}+}$ even achieves comparable performance compared to Oracle, indicating that Ferret effectively enables rapid adaptation. On the other hand, while Ferret$_{\mathrm{M}-}$ shows slightly inferior performance compared to its counterparts, it demands less memory for OCL, as depicted in Fig. 4. This implies that in scenarios where memory is severely constrained, Ferret is the only method capable of learning.

Furthermore, various OCL algorithms are integrated on CORe50/ConvNet in Table 2. It can be observed that Ferret not only mitigates catastrophic forgetting (*i.e.*, increased $tagm$) but also markedly enhances online performance (*i.e.*, increased agm), validating its orthogonality and superiority compared to other OCL frameworks for rapid adaptation.

6.3. Comparisons on Pipeline Parallelism

Table 3 compares $agm_{\mathcal{B}}(\mathcal{A}, T)$ of different pipeline parallelism strategies across 20 different settings to evaluate the performance of Ferret's fine-grained pipeline parallelism

Table 3. Online Accuracy Gain per unit of Memory ($agm_{\mathcal{B}}(\mathcal{A}, T)$) of different pipeline parallelism strategies, where $\mathcal{B}$ is the DAPPLE. Note that "1W", "2W" and "3W" refer to 1, 2 and 3 wave(s) for the Hanayo algorithm, and no gradients compensation is applied to all asynchronous pipeline parallelism strategies for fair comparisons

Setting	Synchronous PP					Asynchronous PP		
	DAPPLE	ZB	Hanayo$_{1W}$	Hanayo$_{2W}$	Hanayo$_{3W}$	Pipedream	Pipedream$_{2BW}$	Ferret$_M$
MNIST/MnNet	$0_{\pm0}$	$6.79_{\pm0.4}$	$2.44_{\pm0.3}$	$5_{\pm0.16}$	$7.12_{\pm0.38}$	$8.16_{\pm0.35}$	$\underline{8.23}_{\pm0.39}$	$\mathbf{8.35}_{\pm0.35}$
FMNIST/MnNet	$0_{\pm0}$	$4.06_{\pm0.36}$	$1.52_{\pm0.45}$	$2.8_{\pm0.65}$	$4.26_{\pm0.64}$	$5.29_{\pm0.53}$	$\underline{5.36}_{\pm0.54}$	$\mathbf{5.48}_{\pm0.53}$
EMNIST/MnNet	$0_{\pm0}$	$2.33_{\pm0.08}$	$0.9_{\pm0.02}$	$1.81_{\pm0.05}$	$2.55_{\pm0.04}$	$2.84_{\pm0.09}$	$\underline{2.99}_{\pm0.07}$	$\mathbf{3.02}_{\pm0.09}$
C10/CNet	$0_{\pm0}$	$1.76_{\pm0.08}$	$0.96_{\pm0.14}$	$1.51_{\pm0.04}$	$1.93_{\pm0.12}$	$2.53_{\pm0.04}$	$\underline{2.78}_{\pm0.06}$	$\mathbf{3.05}_{\pm0.15}$
C100/CNet	$0_{\pm0}$	$0.71_{\pm0.04}$	$0.05_{\pm0.05}$	$0.56_{\pm0.06}$	$0.74_{\pm0.06}$	$0.87_{\pm0.09}$	$\underline{1.11}_{\pm0.06}$	$\mathbf{1.72}_{\pm0.01}$
SVHN/CNet	$0_{\pm0}$	$2.13_{\pm0.32}$	$0.36_{\pm0.15}$	$1.52_{\pm0.23}$	$2.21_{\pm0.26}$	$3.3_{\pm0.24}$	$\underline{3.32}_{\pm0.19}$	$\mathbf{3.61}_{\pm0.16}$
TinyI/CNet	$0_{\pm0}$	$0.18_{\pm0.02}$	$0.03_{\pm0.01}$	$0.19_{\pm0.02}$	$0.19_{\pm0.04}$	$0.26_{\pm0.01}$	$\mathbf{0.52}_{\pm0.03}$	$\underline{0.5}_{\pm0.06}$
CORe50/CNet	$0_{\pm0}$	$4.18_{\pm0.23}$	$1.38_{\pm0.12}$	$3.6_{\pm0.16}$	$4.69_{\pm0.11}$	$\underline{6.03}_{\pm0.17}$	$5.91_{\pm0.22}$	$\mathbf{7.13}_{\pm0.18}$
CORe50-iid/CNet	$0_{\pm0}$	$3.58_{\pm0.02}$	$1.19_{\pm0.18}$	$2.94_{\pm0.02}$	$3.74_{\pm0.07}$	$5.07_{\pm0.13}$	$\underline{5.24}_{\pm0.07}$	$\mathbf{6.18}_{\pm0.05}$
S-MNIST/MnNet	$0_{\pm0}$	$2.94_{\pm0.29}$	$1.33_{\pm0.26}$	$2.97_{\pm0.21}$	$3.69_{\pm0.23}$	$\underline{4.3}_{\pm0.29}$	$4.11_{\pm0.33}$	$\mathbf{4.47}_{\pm0.29}$
S-FMNIST/MnNet	$0_{\pm0}$	$1.56_{\pm0.29}$	$0.91_{\pm0.16}$	$1.48_{\pm0.2}$	$1.89_{\pm0.31}$	$2.06_{\pm0.28}$	$\underline{2.09}_{\pm0.27}$	$\mathbf{2.24}_{\pm0.28}$
S-C10/CNet	$0_{\pm0}$	$0.96_{\pm0.14}$	$0.44_{\pm0.13}$	$1.19_{\pm0.03}$	$1.42_{\pm0.14}$	$\underline{2.21}_{\pm0.08}$	$2.16_{\pm0.05}$	$\mathbf{2.58}_{\pm0.1}$
S-C100/CNet	$0_{\pm0}$	$1.57_{\pm0.05}$	$0.54_{\pm0.14}$	$1.25_{\pm0.12}$	$1.67_{\pm0.12}$	$2.48_{\pm0.12}$	$\underline{2.49}_{\pm0.1}$	$\mathbf{3.49}_{\pm0.06}$
S-SVHN/CNet	$0_{\pm0}$	$0.86_{\pm0.05}$	$0.49_{\pm0.08}$	$0.88_{\pm0.06}$	$1.13_{\pm0.03}$	$1.39_{\pm0.04}$	$\underline{1.58}_{\pm0.06}$	$\mathbf{1.75}_{\pm0.03}$
S-TinyI/CNet	$0_{\pm0}$	$0.27_{\pm0.05}$	$0.08_{\pm0.03}$	$0.14_{\pm0.02}$	$0.22_{\pm0.03}$	$0.29_{\pm0.03}$	$\underline{0.47}_{\pm0.01}$	$\mathbf{0.66}_{\pm0.04}$
CLEAR10/RNet	$0_{\pm0}$	$0.38_{\pm0.13}$	$0.46_{\pm0.08}$	$1.04_{\pm0.06}$	$1.4_{\pm0.04}$	$1.8_{\pm0.06}$	$\underline{1.92}_{\pm0.05}$	$\mathbf{2.12}_{\pm0.05}$
CLEAR10/MoNet	$0_{\pm0}$	$1.03_{\pm0.64}$	$0.65_{\pm0.23}$	$2.31_{\pm0.15}$	$2.65_{\pm0.49}$	$\underline{4.25}_{\pm0.09}$	$3.82_{\pm0.26}$	$\mathbf{5.34}_{\pm0.11}$
CLEAR100/RNet	$0_{\pm0}$	$2.76_{\pm0.1}$	$1.36_{\pm0.22}$	$2.52_{\pm0.21}$	$3.3_{\pm0.23}$	$3.85_{\pm0.2}$	$\underline{3.98}_{\pm0.19}$	$\mathbf{4.24}_{\pm0.22}$
CLEAR100/MoNet	$0_{\pm0}$	$3.11_{\pm0.53}$	$1.26_{\pm0.12}$	$3.03_{\pm0.52}$	$4.24_{\pm0.12}$	$5.66_{\pm0.19}$	$\underline{5.88}_{\pm0.58}$	$\mathbf{7.42}_{\pm0.69}$
Covertype/MLP	$0_{\pm0}$	$0.62_{\pm0.14}$	$0.24_{\pm0.12}$	$0.6_{\pm0.16}$	$0.83_{\pm0.16}$	$\mathbf{0.92}_{\pm0.16}$	$0.82_{\pm0.13}$	$\underline{0.89}_{\pm0.08}$

Table 4.
Online Accuracy differences between Ferret with and without gradients compensation algorithms .

Ferret$_{M+}$				Ferret$_M$			
Step-Aware	Gap-Aware	Fisher	Iter-Fisher	Step-Aware	Gap-Aware	Fisher	Iter-Fisher
$-56.04_{\pm}2.78$	$-14.03_{\pm}1.24$	$\underline{-0.02}_{\pm}0.01$	$\mathbf{0.01}_{\pm}0$	$-43.53_{\pm}2.36$	$-12.11_{\pm}1.03$	$\underline{-0.01}_{\pm}0.01$	$\mathbf{0.02}_{\pm}0$
$-37.75_{\pm}2.17$	$-9.07_{\pm}0.63$	$\underline{-0.02}_{\pm}0.03$	$\mathbf{0.05}_{\pm}0.01$	$-37.74_{\pm}2.53$	$-7.07_{\pm}0.55$	$\underline{-0.01}_{\pm}0.01$	$\mathbf{0.02}_{\pm}0$
$-20.5_{\pm}0.05$	$-4.81_{\pm}0.15$	$\underline{0.01}_{\pm}0.02$	$\mathbf{0.04}_{\pm}0.02$	$-33.36_{\pm}0.13$	$-3.46_{\pm}0.33$	$\underline{0.01}_{\pm}0.01$	$\mathbf{0.04}_{\pm}0.02$
$-10.12_{\pm}0.19$	$-1.71_{\pm}0.43$	$\underline{-0.32}_{\pm}0.07$	$\mathbf{0.25}_{\pm}0.06$	$-9.6_{\pm}0.19$	$-1.22_{\pm}0.39$	$\underline{-0.14}_{\pm}0.3$	$\mathbf{0.42}_{\pm}0.21$
$-8.08_{\pm}0.17$	$-2.17_{\pm}0.05$	$\underline{-0.04}_{\pm}0.08$	$\mathbf{0.13}_{\pm}0.05$	$-5.4_{\pm}0.07$	$-1.04_{\pm}0.04$	$\underline{-0.01}_{\pm}0.07$	$\mathbf{0.1}_{\pm}0.02$
$-14.92_{\pm}0.42$	$-2.63_{\pm}0.04$	$\underline{0.02}_{\pm}0.04$	$\mathbf{0.31}_{\pm}0.2$	$-24.7_{\pm}1.18$	$-2.91_{\pm}0.11$	$\underline{0.22}_{\pm}0.07$	$\mathbf{0.3}_{\pm}0.07$
$-3.72_{\pm}0.16$	$-1.06_{\pm}0.09$	$\underline{-0.01}_{\pm}0.02$	$\mathbf{0.06}_{\pm}0.03$	$-1.32_{\pm}0.17$	$-0.11_{\pm}0.19$	$\underline{0.17}_{\pm}0.19$	$\mathbf{0.35}_{\pm}0.15$
$-23.93_{\pm}0.16$	$-3.27_{\pm}0.05$	$\underline{-0.22}_{\pm}0.11$	$\mathbf{0.1}_{\pm}0.08$	$-33.41_{\pm}0.24$	$-4.18_{\pm}0.34$	$\underline{0.02}_{\pm}0.12$	$\mathbf{0.34}_{\pm}0.07$
$-24.56_{\pm}0.22$	$-3.91_{\pm}0.2$	$\underline{0.22}_{\pm}0.08$	$\mathbf{0.32}_{\pm}0.06$	$-23.77_{\pm}0.3$	$-3.15_{\pm}0.53$	$\underline{0.23}_{\pm}0.12$	$\mathbf{0.39}_{\pm}0.1$
$-26.8_{\pm}3.2$	$-4.21_{\pm}0.21$	$\underline{-0.05}_{\pm}0.02$	$\mathbf{0.03}_{\pm}0.01$	$-46.24_{\pm}1.41$	$-4.82_{\pm}0.48$	$\underline{-0.03}_{\pm}0.01$	$\mathbf{0.02}_{\pm}0$
$-14.43_{\pm}2.83$	$-2.37_{\pm}0.14$	$\underline{0.01}_{\pm}0.02$	$\mathbf{0.03}_{\pm}0.02$	$-46.07_{\pm}4.03$	$-2.1_{\pm}0.33$	$\underline{-0.01}_{\pm}0.01$	$\mathbf{0}_{\pm}0$
$-6.93_{\pm}0.12$	$-1_{\pm}0.14$	$\underline{-0.19}_{\pm}0.09$	$\mathbf{0.1}_{\pm}0.03$	$-8.11_{\pm}0.75$	$-0.99_{\pm}0.24$	$\underline{-0.12}_{\pm}0.12$	$\mathbf{0.23}_{\pm}0.12$
$-14.14_{\pm}0.37$	$-3.05_{\pm}0.1$	$\underline{-0.23}_{\pm}0.18$	$\mathbf{0.38}_{\pm}0.23$	$-12.56_{\pm}0.23$	$-1.92_{\pm}0.09$	$\underline{-0.1}_{\pm}0.19$	$\mathbf{0.24}_{\pm}0.09$
$-5.69_{\pm}0.46$	$-1.18_{\pm}0.12$	$\underline{-0.03}_{\pm}0.03$	$\mathbf{0.05}_{\pm}0.03$	$-12.13_{\pm}0.87$	$-1.27_{\pm}0.09$	$\underline{-0.05}_{\pm}0.02$	$\mathbf{0.03}_{\pm}0.01$
$-3.61_{\pm}0.14$	$-1.01_{\pm}0.05$	$\underline{0.05}_{\pm}0.1$	$\mathbf{0.12}_{\pm}0.1$	$-1.7_{\pm}0.12$	$-0.37_{\pm}0.03$	$\underline{0.08}_{\pm}0.07$	$\mathbf{0.18}_{\pm}0.08$
$-6.26_{\pm}0.18$	$-0.72_{\pm}0.08$	$\underline{0.02}_{\pm}0.06$	$\mathbf{0.14}_{\pm}0.04$	$-11.25_{\pm}0.28$	$-0.52_{\pm}0.05$	$\underline{-0.02}_{\pm}0.04$	$\mathbf{0.08}_{\pm}0.04$
$-18.17_{\pm}0.26$	$-1.7_{\pm}0.05$	$\underline{-0.14}_{\pm}0.32$	$\mathbf{0.5}_{\pm}0.15$	$-20.69_{\pm}0.58$	$-1.88_{\pm}0.46$	$\underline{-0.28}_{\pm}0.37$	$\mathbf{0.36}_{\pm}0.18$
$-17.45_{\pm}0.02$	$-0.38_{\pm}0.54$	$\underline{-0.07}_{\pm}0.04$	$\mathbf{0.65}_{\pm}0.32$	$-24.62_{\pm}0.24$	$\underline{0.18}_{\pm}0.32$	$-0.2_{\pm}0.18$	$\mathbf{1.27}_{\pm}0.21$
$-30.65_{\pm}0.93$	$-0.3_{\pm}0.17$	$\underline{0.63}_{\pm}0.46$	$\mathbf{1.2}_{\pm}0.48$	$-26.85_{\pm}1.22$	$-0.3_{\pm}0.33$	$\underline{0.58}_{\pm}0.78$	$\mathbf{1.03}_{\pm}0.7$
$-9.2_{\pm}1.13$	$-2.8_{\pm}0.1$	$\underline{-0.09}_{\pm}0.05$	$\mathbf{0.05}_{\pm}0.02$	$-10.27_{\pm}0.39$	$-1.52_{\pm}0.14$	$\underline{-0.01}_{\pm}0$	$\mathbf{0.09}_{\pm}0.01$

under memory constraints. Specifically, $\mathcal{B}$ is selected as DAPPLE, and no gradients compensation is applied to any asynchronous pipeline parallelism strategies. Additionally, Hanayo$_{1W}$, Hanayo$_{2W}$ and Hanayo$_{3W}$ are three variants with 1, 2, and 3 waves, respectively.

In general, all asynchronous pipeline parallelism strategies significantly outperform synchronous pipeline parallelism strategies, even ZB, which claims to eliminate pipeline bubbles. This is because synchronous pipeline parallelism strategies, in an effort to achieve higher hardware utilization rates and avoid conflicting model versions, must design complex workflows that stage gradients and update model parameters synchronously, resulting in delays in data processing and wasted data value. Conversely, asynchronous pipeline parallelism strategies process data and update model parameters immediately, thereby minimizing processing latency. Among all asynchronous pipeline parallelism strategies, Ferret$_M$'s fine-grained pipeline parallelism strategy consistently surpasses the others due to its more efficient memory utilization.

To investigate the impact of different pipeline configurations for Ferret, we select five different memory constraints ranging from minimum to maximum to simulate learning under varying memory budgets Fig. 6 shows that Ferret successfully solves Eq. 2 for obtaining optimal pipeline configurations under dynamic environments, scaling effectively as we increase the memory constraint. Specifically, lack of precise control over each pipeline stage to balance between performance and memory footprint prevents competing strategies from scaling well.

6.4. Comparisons on Gradients Compensation

To evaluate the effectiveness of Iter-Fisher, we apply various gradients compensation algorithms to Ferret$_{M+}$ and Ferret$_M$, and compare the final online accuracy gain. The results are shown in Table 4. From the table, we can observe that applying Step-Aware and Gap-Aware algorithms for compensating stale gradients significantly reduces the online accuracy. This is because these algorithms mitigate the gradient staleness problem by simply penalizing the step size of stale gradients, leading to a slow convergence rate when the system is highly paralleled. Although Fisher leverages first-order information for better compensation, it does not consider varying levels of staleness at different stages of pipeline parallelism, resulting in a marginal decrease in accuracy compared to no compensation. On the other hand, Iter-Fisher consistently improves online accuracy across all settings, without requiring manual hyperparameters tuning. This indicates that Iter-Fisher effectively adapts to different levels of staleness in parallelism, and automatically optimizes λ for better compensation, demonstrating its robustness and effectiveness.

7. Conclusion

This paper introduces Ferret, a novel framework designed to boost online accuracy of OCL algorithms under varying memory constraints. Ferret employs a fine-grained pipeline parallelism strategy to adapt to varying distributions of incoming streaming data rapidly. To mitigate the gradient staleness problem in parallel processing, Ferret integrates an iterative gradient compensation algorithm to prevent performance degradation. Additionally, pipelines are automatically scheduled to improve performance under any memory scenario by optimizing a bi-level optimization problem. Extensive experiments conducted on 18 datasets and 5 models confirm Ferret's superior efficiency and robustness compared to existing methods, demonstrating its potential as a scalable solution for adaptive, memory-efficient OCL.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62306198; in part by National Major Scientific Instruments and Equipments Development Project of National Natural Science Foundation of China under Grant 62427820; and in part by the Natural Science Foundation of Sichuan Province under Grant 2024NSFSC1468.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016. 1

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pages 139–154, 2018. 1, 2, 6, 7, 5

[3] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. *Advances in neural information processing systems*, 32, 2019. 1, 2, 6, 7, 5

[4] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019. 1, 2

[5] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. 2024. 1, 2

[6] Jeff Barber, Ximing Yu, Laney Kuenzel Zamore, Jerry Lin, Vahid Jazayeri, Shie Erlich, Tony Savor, and Michael Stumm. Bladerunner: Stream processing at scale for a live view of backend data mutations at the edge. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 708–723, 2021. 1

[7] Saar Barkai, Ido Hakimi, and Assaf Schuster. Gap aware mitigation of gradient staleness. *arXiv preprint arXiv:1909.10802*, 2019. 6, 1

[8] Jock Blackard. Covertype. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C50K5N. 6, 4

[9] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020. 1, 2, 3

[10] Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. New insights on reducing abrupt representation change in online continual learning. *arXiv preprint arXiv:2104.05025*, 2021. 1, 2

[11] Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8281–8290, 2021. 1, 2, 7

[12] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, P Dokania, P Torr, and M Ranzato. Continual learning with tiny episodic memories. In *Workshop on Multi-Task and Lifelong Reinforcement Learning*, 2019. 1, 2, 6, 7, 5

[13] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 3

[14] Yangrui Chen, Cong Xie, Meng Ma, Juncheng Gu, Yanghua Peng, Haibin Lin, Chuan Wu, and Yibo Zhu. Sapipe: Staleness-aware pipeline for data parallel dnn training. *Advances in Neural Information Processing Systems*, 35: 17981–17993, 2022. 6

[15] François Chollet. keras. `https://github.com/fchollet/keras`, 2015. 1

[16] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017. 4

[17] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pages 485–497, 2019. 1, 2

[18] Marcos Dias de Assuncao, Alexandre da Silva Veith, and Rajkumar Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103: 1–17, 2018. 1

[19] Matthias De Lange and Tinne Tuytelaars. Continual prototype evolution: Learning online from non-stationary data streams. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8250–8259, 2021. 1, 2

[20] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021. 3

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[22] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 6, 4

[23] Matthew F Dixon, Igor Halperin, and Paul Bilokon. *Machine learning in finance*. Springer, 2020. 1

[24] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. Dapple: A pipelined data parallel approach

for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 431–445, 2021. 6, 1

[25] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. 1, 2

[26] Enrico Fini, Stéphane Lathuiliere, Enver Sangineto, Moin Nabi, and Elisa Ricci. Online continual learning under extreme memory constraints. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII 16*, pages 720–735. Springer, 2020. 1

[27] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In *International conference on machine learning*, pages 1920–1930. PMLR, 2019. 1, 2

[28] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, 2001. 4

[29] Yasir Ghunaim, Adel Bibi, Kumail Alhamoud, Motasem Alfarra, Hasan Abed Al Kader Hammoud, Ameya Prabhu, Philip HS Torr, and Bernard Ghanem. Real-time evaluation in online continual learning: A new hope. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11888–11897, 2023. 1, 2, 6

[30] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. Imagebind: One embedding space to bind them all. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15180–15190, 2023. 1

[31] Yue Guan, Yuxian Qiu, Jingwen Leng, Fan Yang, Shuo Yu, Yunxin Liu, Yu Feng, Yuhao Zhu, Lidong Zhou, Yun Liang, et al. Amanda: Unified instrumentation framework for deep neural networks. 2023. 1, 2

[32] Nuwan Gunasekara, Bernhard Pfahringer, Heitor Murilo Gomes, and Albert Bifet. Survey on online streaming continual learning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 6628–6637. ijcai. org, 2023. 1

[33] Corentin Hardy, Erwan Le Merrer, and Bruno Sericola. Distributed deep learning on edge-devices: feasibility via adaptive compression. In *2017 IEEE 16th international symposium on network computing and applications (NCA)*, pages 1–8. IEEE, 2017. 6, 1

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[35] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 6

[36] Qinghao Hu, Zhisheng Ye, Meng Zhang, Qiaoling Chen, Peng Sun, Yonggang Wen, and Tianwei Zhang. Hydro:{Surrogate-Based} hyperparameter tuning service in datacenters. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 757–777, 2023. 1, 2

[37] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 385–399, 2020. 1, 2

[38] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019. 1

[39] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 1, 2

[40] Anand Jayarajan, Kimberly Hau, Andrew Goodwin, and Gennady Pekhimenko. Lifestream: a high-performance stream processing engine for periodic streams. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 107–122, 2021. 1

[41] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 463–478, 2017. 6, 1

[42] Jin Kyu Kim, Qirong Ho, Seunghak Lee, Xun Zheng, Wei Dai, Garth A Gibson, and Eric P Xing. Strads: A distributed framework for scheduled model parallel machine learning. In *Proceedings of the Eleventh European Conference on Computer Systems*, pages 1–16, 2016. 1, 2

[43] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6, 4

[44] Young D Kwon, Jagmohan Chauhan, Hong Jia, Stylianos I Venieris, and Cecilia Mascolo. Lifelearner: Hardware-aware meta continual learning system for embedded computing platforms. *arXiv preprint arXiv:2311.11420*, 2023. 2, 3

[45] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015. 6, 3, 4

[46] Yiming Li, Yanyan Shen, and Lei Chen. Camel: Managing data for efficient stream learning. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1271–1285, 2022. 2, 3, 6, 1

[47] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. 6, 7, 1, 5

[48] Zhiqiu Lin, Jia Shi, Deepak Pathak, and Deva Ramanan. The clear benchmark: Continual learning on real-world imagery. In *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 2)*, 2021. 1, 2, 6, 3, 4

[49] Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2023. 6, 1

[50] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on robot learning*, pages 17–26. PMLR, 2017. 6, 3, 4

[51] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017. 1

[52] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, 2022. 1

[53] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys*, 55(12):1–37, 2023. 1

[54] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020. 1, 2

[55] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018. 1, 2

[56] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4316–4336, 2020. 1

[57] MG Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine learning at the network edge: A survey. *ACM Computing Surveys (CSUR)*, 54(8):1–37, 2021. 1

[58] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019. 4, 6, 1

[59] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*, pages 7937–7947. PMLR, 2021. 3, 4, 6, 1

[60] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, page 7. Granada, Spain, 2011. 6, 4

[61] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International conference on machine learning*, pages 16888–16905. PMLR, 2022. 1

[62] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. 2

[63] Alecos Papadopoulos. The information matrix equality: proof, misspecification, and the quasi-maximum likelihood estimator. *Athens University of Economics and Business*, 2014. 4

[64] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 1

[65] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 524–540. Springer, 2020. 1, 2

[66] Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble pipeline parallelism. *arXiv preprint arXiv:2401.10241*, 2023. 6, 1

[67] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020. 1

[68] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017. 1, 2

[69] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 1, 2

[70] Doyen Sahoo, Quang Pham, Jing Lu, and Steven CH Hoi. Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705*, 2017. 1, 2

[71] David Sayce. The number of tweets per day in 2022. 2022. 1

[72] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018. 1

[73] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. 1

[74] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. Sketching linear classifiers over data streams. In *Proceedings of the 2018 international conference on management of data*, pages 757–772, 2018. 1

[75] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1

[76] Ehsan Valavi, Joel Hestness, Newsha Ardalani, and Marco Iansiti. Time and the value of data. *arXiv preprint arXiv:2203.09118*, 2022. 1, 3

[77] Siniša Veseli, John Hammonds, Steven Henke, Hannah Parraga, and Nicholas Schwarz. Streaming data from experimental facilities to supercomputers for real-time data processing. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 2110–2117, 2023. 1

[78] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. 1

[79] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 6, 4

[80] Minhui Xie, Kai Ren, Youyou Lu, Guangxu Yang, Qingxing Xu, Bihai Wu, Jiazhen Lin, Hongbo Ao, Wanhong Xu, and Jiwu Shu. Kraken: memory-efficient continual learning for large-scale real-time recommendations. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–17. IEEE, 2020. 2, 3

[81] Yu Yang, Hao Kang, and Baharan Mirzasoleiman. Towards sustainable learning: Coresets for data-efficient deep learning. In *International Conference on Machine Learning*, pages 39314–39330. PMLR, 2023. 1

[82] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017. 1, 2

[83] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. *arXiv preprint arXiv:2106.01085*, 2021. 1, 2

[84] Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, and Lidong Zhou. Retiarii: A deep learning {Exploratory-Training} framework. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 919–936, 2020. 1, 2

[85] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International conference on machine learning*, pages 4120–4129. PMLR, 2017. 5, 1

[86] Zhen Zheng, Chanyoung Oh, Jidong Zhai, Xipeng Shen, Youngmin Yi, and Wenguang Chen. Hiwaylib: A software framework for enabling high performance communications for heterogeneous pipeline computations. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 153–166, 2019. 1, 2

[87] Yuhao Zhou, Qing Ye, and Jiancheng Lv. Communication-efficient federated learning with compensated overlap-fedavg. *IEEE Transactions on Parallel and Distributed Systems*, 33(1):192–205, 2021. 1

[88] Yuhao Zhou, Mingjia Shi, Yuanxi Li, Yanan Sun, Qing Ye, and Jiancheng Lv. Communication-efficient federated learning with single-step synthetic features compressor for faster convergence. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5031–5040, 2023. 1

[89] Yuhao Zhou, Minjia Shi, Yuxin Tian, Qing Ye, and Jiancheng Lv. Defta: A plug-and-play peer-to-peer decentralized federated learning framework. *Information Sciences*, 670:120582, 2024. 1

[90] Yuhao Zhou, Yuxin Tian, Mingjia Shi, Yuanxi Li, Yanan Sun, Qing Ye, and Jiancheng Lv. E-3sfc: Communication-efficient federated learning with double-way features synthesizing. *IEEE Transactions on Neural Networks and Learning Systems*, (99):1–15, 2025. 1