Patching in Order: Efficient On-Device Model Fine-Tuning for Multi-DNN Vision Applications

Zhiqiang Cao, *Student Member, IEEE,* Yun Cheng, *Member, IEEE,* Zimu Zhou, *Member, IEEE,* Anqi Lu, *Student Member, IEEE,* Youbing Hu, *Student Member, IEEE,* Jie Liu, *Fellow, IEEE,* Min Zhang, *Member, IEEE,* and Zhijun Li, *Member, IEEE*

Abstract—The increasing deployment of multiple deep neural networks (DNNs) on edge devices is revolutionizing mobile vision applications, spanning autonomous vehicles, augmented reality, and video surveillance. These applications demand adaptation to contextual and environmental drifts, typically through fine-tuning on edge devices without cloud access, due to increasing data privacy concerns and the urgency for timely responses. However, fine-tuning multiple DNNs on edge devices faces significant challenges due to the substantial computational workload. In this paper, we present PatchLine, a novel framework tailored for efficient on-device training in the form of fine-tuning for multi-DNN vision applications. At the core of PatchLine is an innovative lightweight adapter design called patches coupled with a strategic patch updating approach across models. Specifically, PatchLine adopts drift-adaptive incremental patching, correlation-aware warm patching, and entropy-based sample selection, to holistically reduce the number of trainable parameters, training epochs, and training samples. Experiments on four datasets, three vision tasks, four backbones, and two platforms demonstrate that PatchLine reduces the total computational cost by an average of 55% without sacrificing accuracy compared to the state-of-the-art.

Index Terms—Patch, on-device, model adaptation, multi-DNN.

1 Introduction

Mobile vision systems are increasingly adopting *multiple* deep neural networks (DNNs) on edge devices for comprehensive understanding of complex environments [1], [2], [3]. For instance, autonomous vehicles recognize road signs, detect pedestrians, and mark lane boundaries using a mix of DNNs for tasks like image classification [4], object detection [5], and semantic segmentation [6]. Augmented Reality (AR) applications leverage image classification to identify real-world objects, object detection to overlay virtual entities, and semantic segmentation to integrate virtual elements with the physical environment [7]. Safety monitoring system in factories utilizes image classification to detect dangerous worker movements, object detection to determine whether workers are appropriately attired, and semantic segmentation to identify when a worker enters a hazardous area [8]. Similarly, smart wearable devices like fitness bands and smartwatches deploy a multi-DNN framework to detect visual markers on the skin through image classification and analyze bio-metrics like sweat on the skin's surface via semantic segmentation [9].

- *Z. Cao, Y. Hu, A. Lu, J. Liu, M zhang and Z. Li are with the Faculty of Computing, Harbin Institute of Technology, 150006, Harbin, China. (E-mail: {zhiqiang_cao,luanqi,youbing}@stu.hit.edu.cn; {lizhijun_os, zhangmin2021, jieliu}@hit.edu.cn)*
- *Y. Cheng is with Swiss Data Science Center, 8000 Zurich, Switzerland. (E-mail: chengyu@ethz.ch)*
- *Z. Zhou is with the School of Data Science (SDSC), City University of Hong Kong, Hong Kong, China. (E-mail: zimuzhou@cityu.edu.hk)*

This work is partly supposed by the National Key R&D Program of China under Grant NO.2023YFB4503100. This work is also partly supported by NSFC 62072137. Zimu Zhou's research is supported by CityU APRC grant (No. 9610633). (Corresponding author: Zhijun Li, Yun Cheng)

These DNNs are typically trained on the cloud and then deployed to edge devices for inference. Due to the difference in data distribution between training and inference time, *i.e.*, *data drift*, these models often suffer from accuracy degradation [10], [11]. Such data drift is not uncommon because many mobile vision systems operate in dynamic contexts, which increase the frequency of encountering input data from the unseen distributions. Accordingly, model fine-tuning is necessary to resume the inference accuracy, and *on-device model fine-tuning* is preferable to cloud offloading, driven by increasing data privacy concerns and the urgency for timely responses [12], [13]. Take safety monitoring as an example, in case of varying light and weather conditions, *e.g.*, from sunlit to foggy, the onboard DNNs must rapidly adapt to environmental changes, often without cloud connectivity [14], [15].

To counteract data drift, we resort to *fine-tuning* pre-trained DNNs on newly acquired data, a simple yet versatile strategy [16], [17]. However, fine-tuning modern DNNs on edge devices with limited resources is challenging. For example, training classical DNNs such as MobileNetv2 [18], YOLOv4 [19], and DeepLabv3+ [20] for image classification, object detection, and semantic segmentation respectively demands peak memories of 4523 MB, 5911 MB, 6417 MB and the computational workload of each batch is 77.34 GFLOPs, 266.30 GFLOPs, 601.44 GFLOPs (batch sizes: 16, 16, 8). Such memory and computational demands would easily exceed the capacities of commodity edge devices, such as the smart phone [21] and NVIDIA Jetson TX2 [22].

There are different tactics for on-device DNN adaptation or training. Some reduce the amount of trainable model parameters by selecting important parameters [23], [24] or designing lightweight adapters [25]. Others accelerate

model training convergence by learning rate scheduling [26] or warm starting [27]. There are also orthogonal approaches such as reducing the training data by selecting the most informative samples [13]. We focus on *lightweight adapter design* for multi-DNN vision applications. The reasons are two-fold. *(i)* Prior studies have shown the effectiveness of solely updating adapters to combat data drift in vision tasks [12], [28]. *(ii)* By updating adapters and freezing the original model parameters, we can mitigate catastrophic forgetting during model continuous adaptation [29].

While a straightforward adoption of existing on-device DNN training methods to individual DNNs is feasible, there are two unexplored opportunities to elevate the adaptation *efficiency* of multi-DNN vision systems.

- *Share adapter architectures among models*. Models in multi-DNN setups often employ the same backbone architecture (with different parameters) as feature extraction, followed by specialized decoders for downstream tasks. Since many visual drift types are more appropriately adjusted at the backbone level, we propose *task-agnostic* adapters rather than *model-specific* ones. This would notably simplify the adapter design for multi-DNN vision applications.
- *Transfer adapter parameters across models*. The models in multi-DNN vision applications are correlated since they all process images and employ the same backbone architecture. Consequently, we may harness the trained adapter of one model as a *warm start* (initialization) for training adapters in subsequent models. This would reduce the required training epochs for subsequent models.

Nevertheless, it poses distinct challenges to integrate these opportunities into a functional solution.

- *Challenge 1: How to craft task-agnostic yet drift-adaptive adapter architectures?* A static adapter architecture risks introducing unnecessary adaptation overhead in terms of computation, memory, and latency.
- *Challenge 2: How to decide the sequence of adapter parameter transfers across models?* Model correlations differ across tasks, and transferring parameters from less correlated tasks might hinder model convergence.

To address these challenges, we introduce PatchLine, an efficient on-device model fine-tuning framework tailored for multi-DNN vision applications. Given multiple pre-trained models that share the backbone architecture but diverge in downstream tasks [30], [31] (*e.g.*, image classification, object detection, semantic segmentation), PatchLine fine-tunes them by carefully designed adapters known as *patches*. Specifically, PatchLine offers:

- *Drift-adaptive Incremental Patching*. Inspired by previous studies [28], [32], we design base patches as tiny residual module attached to convolution groups within the backbone. A patch predictor, trained through extensive design exploration (what, where, and how many to patch). We add basic patches in an incremental fashion from the input layers based on the drift severity measured by the MMD (Maximum-Mean-Discrepancy) [33] metric.
- *Correlation-aware Warm Patching*. Instead of arbitrary patch initialization, we employ a dedicated patching ordering based on the inter-model correlations. Intriguingly, this patching sequence, determined offline, remains robust against diverse data drift scenarios.

PatchLine also employs standardized techniques such as entropy-based sample selection to further reduce the overall fine-tuning workload. Evaluations on three datasets and two backbones show that entropy-based sample selection reduces the number of training samples by 9%-14%.

Our main contributions are summarized as follows.

- To the best of our knowledge, this is the first on-device fine-tuning scheme for multi-DNN vision systems. We exploit two less unexplored opportunities in on-device DNN training: *(i)* handling drifts in vision tasks at backbone level for model-agnostic adapter design; and *(ii)* harnessing inter-model correlation for warm adapter initialization.
- We propose PatchLine, a holistic solution to improve the adaptation efficiency of multi-DNN vision applications by reducing trainable parameters, training epochs, and the training samples via drift-adaptive incremental patching, correlation-aware warm patching, and entropy-based sample selection.
- Extensive evaluations on four different drift datasets, four common backbones and three basic vision tasks demonstrate PatchLine reduces the total computational cost by an average of 55% without sacrificing accuracy compared to the state-of-the-art model adaptation framework (LST [32]).

2 Related Work

2.1 Adaptation to Data Drift

There is extensive research on model adaptation to data drifts [11], [34], [35], including the catastrophic forgetting during continuous adaptation [29], training resource allocation [36], trigger conditions for model updates [34], and how to obtain supervision information for new sampling data [13]. It is also an important topic in mobile and ubiquitous computing. Khani *et al.* [11] adaptively sample video frames by continuously monitoring environmental changes in real-time to improve the model adaptation accuracy. Xu *et al.* [37] dynamically adjusts model parameters on the IoT devices through local inference on drift data. Chauhan *et al.* [38] applies model adaptation to time series and mitigates data drift for behavior-based use authentication. Gan *et al.* [13] handle accuracy degradation in a constantly changing environment by presenting a model adaptation framework for autonomous driving. Kong *et al.* [15] continuously update the lightweight model at the edge device to improve the video surveillance accuracy in adverse environments. Ekya *et al.* [36] design a scalable resource scheduler for joint model training and inference on edge servers. Less GPU resources used without compromising accuracy.

Different from the above methods, the principal objective of our work is to significantly reduce the computational cost in multi-DNN adaptation for vision tasks.

2.2 On-device DNN Training

On-device DNN training is motivated by the need for privacy-preserving model customization and personalization with local user data [39], [40]. The primary bottleneck of DNN training on resource-constrained devices is memory [23], [28]. For memory-efficient training, earlier studies [16], [41] suggest updating only the last few layers, which leads to poor accuracy, especially on datasets with notable drifts [28]. p-Meta [23] automatically identifies adaptation-critical parameters at runtime to minimize the memory footprint. TinyTL [28] only updates the bias rather than weights, and introduces the lite residual module to improve the model capacity. To speed up on-device training, ElasticTrainer [42] allows fully elastic selection of tensors to adapt to the runtime need of training. Mercury [43] focuses on samples that provide more important information in each training iteration. Taking inspiration from Mercury, we propose an entropy-based sample filtering method that removes redundant data without compromising training accuracy. To improve model accuracy, LST [32] proposes a ladder side network, *i.e.*, a small and separate network that takes intermediate activations as input via shortcut connections (called ladders) from the backbone.

Our patching shares similar principles as the partial parameter update strategies [12], [28]. But we focus on visual tasks adopting a "backbone + decoder" architecture, resulting in a more structured design (residual layers per convolution group). The patch architecture is inspired by LST [32]. However, we do not add patches to all convolution groups, but adaptively change the number of patches based on the drifting data, without compromising accuracy. More importantly, we harness the task correlations to reduce the training workload across models, which is largely unexplored in previous on-device DNN training studies.

2.3 Multi-task Learning in Mobile Applications

Multi-task learning exploits the correlation between tasks, allowing multiple tasks to be trained together by sharing structure, which improves generalization [44], [45]. It is widely adopted in ubiquitous computing applications for effective model training across sensing modalities [46], devices [47], users [48], tasks [49] etc. For example, Roy *et al.* [1] improve the accuracy of vehicle detection by joint learning of data from multiple sensing modalities, including image, radar, acoustic, and seismic data. Sugarmate [48] designs grouped input layers, together with the adoption of a deep RNN model, to build blood glucose models for the general public based on limited personal measurements from single-user and grouped-users perspectives. Dai *et al.* [50] introduce a specialized Multi-Task Learning (MTL) model designed for randomized controlled experiments. LU *et al.* [47] apply MTL in the field of depression detection, achieving more precise depression detection through joint learning from data collected from multiple devices.

Our work differs from multi-task learning in the problem settings. Instead of training a multi-task network from scratch, we aim at efficient fine-tuning of multiple separately pre-trained models.

2.4 Efficient Multi-DNN Inference

This thread of research accelerates the execution of multiple pre-trained DNNs at different optimization levels. For example, MTZ [51] enforces cross-model weight sharing for storage saving. PAM [52] integrates network pruning into model merging, offering a heuristic approach to minimize the computational cost when executing any task subset. Wang *et al.* [3] propose a speculative multi-model inference framework to improve the accuracy on complex scenarios. MTS [53] proposes a graph rewriter for efficient multi-task inference with weight-shared DNNs. Heimdall [54] coordinates both DNNs and non-DNNs with mobile GPUs. Band [55] further optimizes the execution of multiple DNNs on heterogeneous computational resources. Layerweaver [56] reduces the temporal waste of computational resources by interweaving layer execution of multiple different models with opposing characteristics: compute-intensive and memory-intensive. LEO [57] proposes a scheduling framework to distribute the sensor processing tasks across the broader range of heterogeneous computational resources of mobile phones (CPU, co-processor, GPU and the cloud).

Our work is inspired by the rationales in these studies, yet we exploit task relatedness among multiple pre-trained models for training, which is more challenging and resource-demanding than inference.

3 Problem Scope

3.1 Multi-DNN Adaptation

Given N models (deep neural networks) are pre-trained for different vision tasks on source domain dataset D_s, we aim to fine-tune the N models on drift dataset D_{drift} to resume model accuracy at low training overhead. We consider the following problem settings (see Fig. 1).

- *Drift Types And Detection.* The source dataset D_s and the drifted dataset D_{drift} differ in environment conditions (*e.g.*, fog, rain, snow), pose and appearance (*e.g.*, clothes and angles for objects in the person class) etc. These drift types are common in applications such as safety monitoring [8] and autonomous vehicles [13]. Data Drift can be quantified and detected through the following steps: *(i)* Obtain feature vector sets F_{source} and F_{drift} for source domain and drift data from intermediate layers of pre-trained visual models. *(ii)* Use Maximum Mean Discrepancy (MMD) [33] to compare two distributions:

$$\text{MMD}^2(F_{\text{source}}, F_{\text{drift}}) = \left\| \frac{1}{n}\sum_{i=1}^{n}\phi(x_i) - \frac{1}{m}\sum_{j=1}^{m}\phi(x_j) \right\|_{\mathcal{H}}^2 \quad (1)$$

  where ϕ is a kernel function mapping to a reproducing kernel Hilbert space (RKHS). A large positive MMD indicates data drift.
- *Model Architectures*. We consider models adopting a "backbone + decoder" architecture for diverse visual tasks. Models in multi-DNN setups often employ the same backbone architecture (with different parameters) as feature extraction, followed by specialized decoders for down-stream tasks. For example, an autonomous vehicle may adopt MobileNetv2 for image

TABLE 1
Static memory, peak dynamic memory, and computation of each batch on example datasets. Batch sizes are 16 for image classification, 16 for object detection, and 8 for semantic segmentation.

Application	Model/Benchmark	Static Memory (MB) Model/Sample	Peak Dynamic Memory (MB)	Training Computation of Each Batch (GLOPs)
Image Classification	MobileNetv2/Core50 [61]	9.5/1.4	4523	77.34
Object Detection	YOLOv4-MobileNetv2 /Core50	49.2/1.4	5911	266.30
Semantic Segmentation	DeepLabv3+-MobileNetv2 /Core50	23.9/1.4	6417	601.44

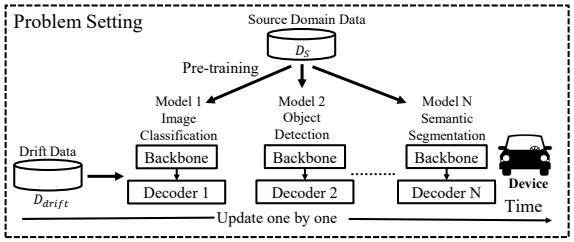


Fig. 1. An illustration of problem setting.

classification, YOLOV4-MobileNetv2 for object detection, and DeepLabv3+-MobileNetv2 for semantic segmentation, respectively.

- *Efficiency Metrics*. We mainly adopt the overall computational cost (in FLOPs) to measure the training overhead. It can also imply the training latency [58], which is crucial for rapid adaptation to drift data.
- *Sequential Adaptation*. We assume sequential adaptation of the N models. This is because parallel adaptation may overwhelm the memory budget on edge devices. As shown in Table 1, simultaneously training MobileNetv2 and YOLOv4-MobileNetv2 requires 10.4 GB of memory, whereas the maximum memory of a smartphone [59] or an NVIDIA Jetson TX2 [22] is within 8 GB. Furthermore, task-level parallelism, where each model corresponds to one task, is not well supported by many deep learning frameworks [53], [60].

3.2 Optimization Opportunities

Adaptation Workload Decomposition. Following the setups in Sec. 3.1, we can roughly estimate the workload for multi-DNN adaptation as follows. For simplicity, we assume standard mini-batch gradient descent is adopted. The computation C_i of model i can be calculated as

$$C_i = E_i \times (b_i \times n_i) \times (F_i + B_i) \tag{2}$$

where E_i, b_i, n_i denote the number of epochs, batch size and the number of batches, respectively. F_i and B_i represent the computation of a single forward and backward pass. Since we assume the N models are updated sequentially, the total computational workload as follow:

$$C = \sum_{i=1}^{N} C_i = \sum_{i=1}^{N} (E_i \times (b_i \times n_i) \times (F_i + B_i)) \tag{3}$$

Opportunities to Decrease Adaptation Workload. Although Eq. (3) shares the same form as the computational cost in vanilla training, adapting pre-trained models brings distinct opportunities to lower the computational overhead.

- *Reduce single-pass training cost $(F_i + B_i)$ via partial model update*. The computational cost of forward and backward propagation is proportional to the number of parameters updated in the model. In model adaptation, it is viable to only train a subset of important parameters to handle data drifts.
- *Reduce training epochs E_i via model parameter transfer*. Since the N models are correlated, there is potential to reuse certain trained parameters of a highly relevant model without retraining from scratch.
- *Reduce training samples $(b_i \times n_i)$ via sample selection*. It is only necessary to train the models on data samples critical to contextual changes, rather than on the entire dataset to learn the visual tasks.

We aim at a comprehensive solution that optimizes the adaptation workload from all the three aspects above for multi-DNN vision applications, as explained below.

4 PatchLine Overview

This section presents an overview of PatchLine, an efficient on-device model fine-tuning framework for multi-DNN vision applications. PatchLine attaches lightweight adapters known as *patches* to each model, and only fine-tunes these patches in a strategic *sequence* without altering the original model parameters, to achieve high accuracy on drifted data with low adaptation cost.

Design Rationales. As mentioned in Sec. 3.2, PatchLine systematically decreases the total computation of multi-DNN adaption in Eq. (3) via three strategies.

- **Patching Scheme**. PatchLine decreases the single-pass training cost $(F_i + B_i)$ by crafting patches dedicated to drift adaptation in multi-DNN vision applications. By freezing the original model parameters and only updating the patches, the single-pass training cost is estimated as $(F_i+P_F+P_B)$, where P_F and P_B are the patch's computations in the forward and backward pass. We propose *task-agnostic* base patches and *drift-adaptive* patching schemes where $P_F + P_B \ll B_i$ even on severely drifted data.
- **Patch Training Order**. Instead of training the patches of the N models independently from scratch, PatchLine intelligently transfers the trained patches of one model to another, such that the training epochs of model i reduce from E_i to E_i', where $E_i' < E_i$ expect for the first model. Such parameters transfer as warm-starts is viable since *(i)* the models are correlated and *(ii)* the tasks share the same base patch

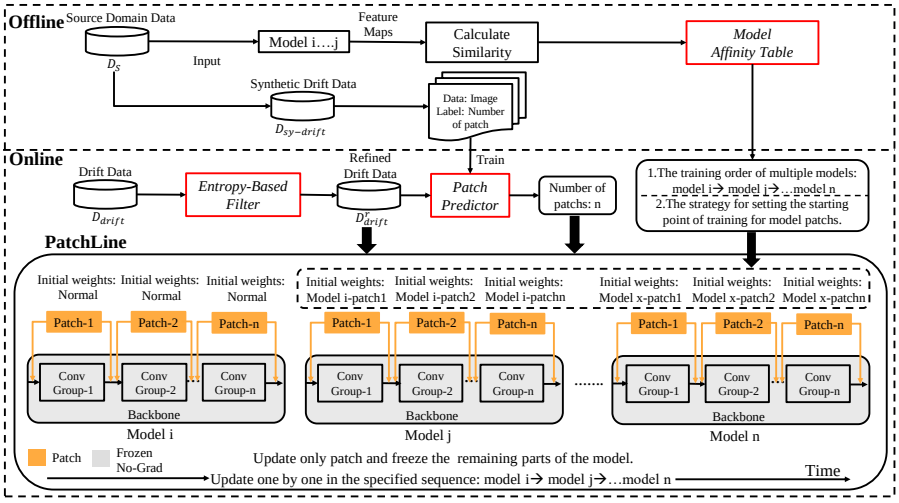


Fig. 2. Illustration of PatchLine, where training multiple models in a specific order, with the acceleration of convergence facilitated by the transfer of patch weights between models.

architectures. PatchLine employs a *correlation-aware* patch update ordering to maximize patch parameter reuse during training.

- **Sample Selection**. Though not a main contribution, PatchLine utilizes a lightweight criterion to select important samples for patch training, which reduces the training samples of model i from $b_i \times n_i$ to $b_i \times n_i'$, where $n_i' < n_i$. We keep the batch size b_i fixed since it may affect the convergence of standard DNN optimizers such as mini-batch gradient descent. Such optimizations are beyond the scope of this work.

With the above optimizations, the total computation of multi-DNN adaption is reduced to

$$C_{patch} = \sum_{i=1}^{N} (E_i' \times (b_i \times n_i') \times (F_i + P_F + P_B)) \quad (4)$$

where $E_i' \leq E_i$, $n_i' < n_i$, and $F_i + P_F + P_B < F_i + B_i$. Empirically, we observe PatchLine reduces the number of epochs E_i by up to 47%, number of training samples n_i by up to 17% , and the single-pass training overhead $(F_i + B_i)$ by up to 67% (see Sec. 6.2).

Functional Modules. PatchLine consists of three modules: *drift-adaptive incremental patching*, *correlation-aware warm patching*, and *entropy-based sample selection*, which decreases the trainable parameters, training epochs, and the training samples, respectively.

- **Drift-adaptive Incremental Patching (Sec. 5.1)**. This module incrementally adds base patches to the backbones of each model. Base patches can be attached as residual connections per convolution group starting from the input end. The number of base patches is decided by a patch predictor according to the data drift level. All the models share the same base patch architectures and the number of base patches.
- **Correlation-aware Warm Patching (Sec. 5.2)**. This module accelerates patch training by recursively transferring trained patch parameters of one model to another as its warm initialization. The patch training sequence is derived from a model affinity table trained offline.
- **Entropy-based Sample Selection (Sec. 5.3)**. This module picks samples notably deviate from the source domain for patch training. It adopts a lightweight entropy-based criterion on the image classification model output score to select important samples without extra computation overhead.

Operational Workflow. As illustrated in Fig. 2, PatchLine works in two phases.

- **Offline Phase**. We first generate the model affinity table for correlation-aware warm patching. Specifically, we perform inference with N models on the same image sampled from the source dataset D_s, and assess the inter-model correlation by comparing the similarity between the feature maps from the same convolution groups across models. We then train the patch predictor for drift-adaptive incremental patching. Concretely, we synthesize diverse drifted data $D_{sy-drift}$ from D_s, and train a neural network that takes the drift level as input, and outputs the number of patches to add.
- **Online Phase**. Given the actual drifted dataset D_{drift}, we first remove unimportant samples by the entropy based filter. The remaining samples are fed into the patch predictor to determine the number of patches to insert. After attaching the patches to each model, we train the patches of the N models on the filtered samples based on the sequence given by the model affinity table. That is, the trained patches of one model are used as initialization of the subsequent models to accelerate model convergence.

5 PatchLine Design

We now explain the details of how PatchLine reduces trainable parameters, training epochs, and the training

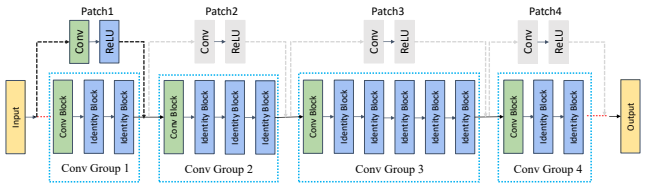


Fig. 3. Illustration of patching scheme (ResNet50 backbone).

samples via drift-adaptive incremental patching (Sec. 5.1), correlation-aware warm patching (Sec. 5.2), and entropy-based sample selection (Sec. 5.3).

5.1 Drift-adaptive Incremental Patching

This subsection introduces *what*, *where*, and *how many* patches to insert into the models. The objective is to devise *task-agnostic* and *drift-adaptive* patching scheme. Fig. 3 shows an example, patching on the ResNet50 backbone. Note that the backbone architecture can be used for diverse visual tasks such as image classification, object detection, and semantic segmentation.

5.1.1 Base Patch Architecture Design

Inspired by residual networks [62], we employ a base patch consisting of a single convolution layer followed by an activation layer and attach it as a residual module to the original model, since single-layer side-path convolutions demonstrated high learning capabilities [28], [32]

The kernel size of the convolution layer within the patch is determined by the backbone structure. To ensure the single-pass training computation of patches is lower than that of the original model, *i.e.*, $F_i + P_F + P_B < F_i + B_i$ (see Sec. 4), we adopt small kernel sizes such as 1x1 and 3x3, and apply Algorithm 1 to check whether the constraint is met. Since the ratio of backward pass operations to forward pass operations is 2:1 [58], we only need to check whether the ratio R (the output of Algorithm 1) of P_F to F_i is within 2/3. For the four backbones (VGG16 [63], ResNet50 [62], MobileNetv2, and Swin-Transformer-Tiny [64]) used in our evaluation, we choose 1x1 convolution layers for the base patch, which easily satisfies the ratio constraint. The parameter count of the patch modules is significantly lower than that of the original models. For instance, VGG16, ResNet50, MobileNetv2, and Swin-Transformer-Tiny have parameter counts of 134.29 M, 23.52 M, 2.23 M, and 28.27 M, respectively, while the patches only introduce 0.43 M, 2.78 M, 0.08 M, and 0.98 M parameters. That is, the patches only increase the parameter count by an average of 4.8%.

Note that we only add patches to the backbone rather than the entire model. This would allow model-agnostic base patch design, since different models often adopt the same backbone architecture (with different parameters) in multi-DNN vision applications.

5.1.2 Extending Base Patches

A single base patch is insufficient to handle severe drift data. There are primarily two ways of patch extension: *(i)* insert multiple patches parallel to convolution groups of the backbone (Fig. 4 (a)); and *(ii)* increase the number of convolution layers within a patch (Fig. 4 (b)). We adopt the former strategy because distributing multiple patches over different convolution groups allow adaptation at diverse scales, while previous studies [12], [28] have already showed the capability of single-convolution residual modules, as mentioned in Sec. 5.1.1.

Algorithm 1: Patch Dimension Checking Algorithm

INPUT: S_m: The structure of model, S_p: The structure of patch, H, M: The size of train data
OUTPUT: R: The forward computation of patch ratio to the forward computation of model.
ALGORITHM:
  CALCULATE C_m**: THE FORWARD COMPUTATION OF MODEL**
  for layer in S_m:
    $C_{layer}, H', M' = F(layer, H, M)$
    $H = H', M = M'$
    $C_m \leftarrow C_m + C_{layer}$
  CALCULATE C_p**: THE FORWARD COMPUTATION OF PATCH**
  for patch in S_p:
    $C_{patch}, H', M' = F(patch, H, M)$
    $H = H', M = M'$
    $C_p \leftarrow C_p + C_{patch}$
  CALCULATE R**:**
    $R \leftarrow C_p / C_m$
  Return R

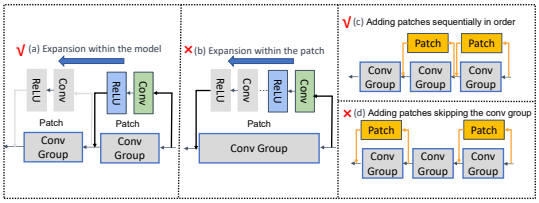


Fig. 4. Illustration of potential ways to extend base patches.

Given multiple base patches, an immediate follow-up question is: which convolution groups to attach them to. We sequentially attach base patches to each convolution group starting from the input end, as shown in Fig. 4 (c). We insert patches from the input because the shallow layers of the network is important as an input to subsequent conv groups. We add patches without skipping convolution groups like in Fig. 4 (d) to allow feature co-adaptation on successive layers, which is essential for effective transfer learning [65]. We empirically compare continual patching and skip patching in Sec. 6.3.

5.1.3 Deciding Number of Patches via Patch Predictor

Since we freeze the original model's weights during training and only update the patches, the model's capacity is proportional to the number of patches. Intuitively, the larger data drift, the more patches are needed. We automatically determine the number of patches according to the drift level via a patch predictor.

The patch predictor is a neural network that takes drifted data as input and outputs the number of patches for a given

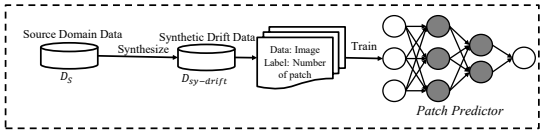


Fig. 5. Illustration of patch predictor, where source dataset D_s is utilized to train a patch predictor.

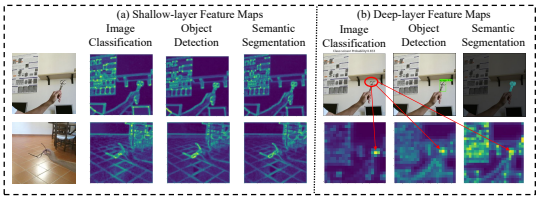


Fig. 6. Example feature maps at shallow (a) and deep (b) convolution groups in the backbones of models for image classification, object detection, and semantic segmentation. From (a), the shallow layers of the three tasks are similar and mainly extract edge features. From (b), the deep layers of the three tasks extract distinct features for the "scissors" class. Importantly, the deep layers of the three tasks still share part of the feature maps despite their differences.

backbone architecture. It is implemented using VGG16 as the feature extraction network, followed by 3 fully connected layers. We only randomly pick part of the drifted dataset (10% by default) to fed into the patch predictor. The output dimension of the patch predictor is backbone-specific. For example, ResNet50 has 4 convolution groups, and thus the output dimension is 5 (0 to 4 patches added).

Fig. 5 illustrates the training of the patch predictor. Note that we do not train a different patch predictor per model because they share the same backbone structure. We use the simplest visual task *e.g.*, image classification, to train the patch predictor. The most critical step is to generate the training data for the patch predictor. Specifically, we first synthesize drift data $D_{sy-drift}$ with various degrees of drift as in [13] using source domain data D_s. We adopt MMD (Maximum-Mean-Discrepancy) to measure the drift level of synthesized data $D_{sy-drift}$. In total, we generate 18 datasets with MMD ranging from 0.0579 to 1.5670 from the source domain dataset D_s. The labels for the synthetic data $D_{sy-drift}$ are manually annotated as follows. First, we generate labels by attaching different numbers of patches to the backbone and training them with various drifts. Then, we label the drift data where the label represents the minimal number of patches for the model to adapt to such drift. For example, the source domain data D_s is labeled as 0 because no patch is necessary to adapt to the source domain data. After generating the training dataset, we train the patch predictor in a standard supervised manner.

Discussions. We make the following notes on the drift-adaptive incremental patching scheme.

- *How is the base patch design different from previous studies?* Previous research has focused on how to efficiently fine-tune a small subset of parameters inside the whole model, e.g. p-meta [23] only updates the parameters of key layers and channels, BitFit [66] only updates the bias. There are also some works on inserting some lightweight modules into the model, but they all design the base module from the perspective of accuracy [25], number of parameters [67], or memory [12], whereas we look at it from the perspective of computational overhead. And the previous base module design only needs to be for a single task, *e.g.*, DCCL [67] only applies to the user recommendation task, whereas our base patch needs to apply to multiple vision tasks.
- *How to train a single patch predictor shareable across tasks?* Firstly, multiple tasks need to share the same backbone structure, which is a prerequisite for being able to use a single patch predictor. This is because different backbone structures have different numbers of convolution groups and adaptability to drift data. Then for multiple tasks, the most basic task (*e.g.*, the basic visual task image classification) needs to be selected to be trained on multiple artificial drift datasets. Based on the training results, these artificial data are labeled and used to train the patch predictor. A patch predictor is comprised of a classification model whose output layer dimension is based on the number of conv groups used in the selected backbone.

5.2 Correlation-aware Warm Patching

This subsection explains how to strategically transfer patch parameters between models to accelerate their training convergence. The basic idea is to utilize the trained patches of the models positioned at the front of the training order as the initialization for training the patches of the models at the back of the training order. The training order is determined by the model affinity table derived from the task relatedness.

5.2.1 *Feasibility of Cross-model Patch Transfer*

Although we assume the N models sharing the same backbone and patch architecture, their parameters differ across models. However, it is viable to transfer parameters across models from the following observations.

- *Observation 1*: The convolution groups of the backbones for different tasks have similar functionalities, particularly in the shallow layers. Previous studies [65] validate that shallow convolution groups extract simple features such as edges, colors, and textures, while deep convolution groups focus on more complex features in image classification. We have similar observations in the *backbones* of *different visual tasks e.g.*, image classification, object detection, and semantic segmentation (see Fig. 6).
- *Observation 2*: Since patches are attached as residual modules per convolution group, they inherently align with the functionality of the corresponding convolution group [12], [25]. This is because the function of the convolution group in the model is related by where it is in the position and the scale of the input, whereas the residual modules has the same input and position as the convolution group.

Observation 1 has three implications. *(i)* The feature maps may indicate the functional differences in the backbones

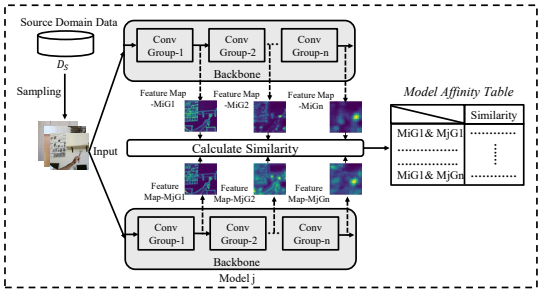


Fig. 7. Illustration to calculate model affinity, where source dataset D_s is used to measure the affinity between models.

across visual tasks. *(ii)* We may share or transfer parameters across the corresponding convolution groups in the backbones of different visual tasks. *(iii)* Even parameter sharing at deep layers is meaningful because there is still overlap among the feature maps in the deep layers of different visual tasks. From *Observation 2*, these implications also apply to the patches, which shows the potential of cross-model patch parameter sharing or transfer.

5.2.2 Efficient Patch Transfer as Ordered Initialization

As mentioned in Sec. 4, we recursively transfer the trained patch of one model to another as initialization to accelerate patch training. The overall patch training workload is minimized if the patch to be trained is initialized from the patches of the most related task. Our goal is to determine the training sequence based on the inter-task correlation.

We assess the task correlation from their similarities in the feature maps, as shown in Fig. 7. Specifically, we calculate the similarity of feature maps per convolution group in the backbones for each model pair, and store the similarity in a model affinity table. The feature maps are generated by inferring on a sampled version of source dataset D_s without accounting for drifts. This is because the functionalities of convolution groups in the backbone are likely to stay unchanged despite contextual drifts such as weather and pose. Therefore, the model affinity table is a one-off offline effort. Since only the relative similarity ranking rather than the absolute value is needed, we explore various efficient similarity metrics such as Mean Hashing [68], Perceptual Hashing [69], and Three-Channel Histogram [70], and opted for Perceptual Hashing since it can accurately estimate the similarity ranking at low computational overhead.

During the online phase, we calculate the average affinity of convolution groups from the model affinity table and the number of patches. For instance, assume we need 3 patches according to the patch predictor. Then we compute the mean likeness of the first three patches of two models to decide the affinity between these two models based on the model affinity table. The affinities between models are calculated pairwise. The model with the highest total affinity is updated first, followed by the model with the second highest total affinity, and so on. The first model is initialized randomly, while subsequent models are initialized from the one with the highest affinity relationship. If two models have the same total affinity, they can be selected randomly.

Discussions. We make the following notes on the correlation-aware warm patching.

- *Why transfer patches at the model level?* The feature co-adaptation also exists to successive patches of the same model [65]. Removing the conv groups exposes that the patches interconnect into a lightweight multi-layer neural network, so that the patch-to-patch relationship is equivalent to successive DNN layers. So inserting a mix of patches from different models is not conducive to knowledge transfer.
- *Why does static training ordering work for diverse data drifts?* Firstly, the order in which we train tasks is determined by their affinity correlation, an objective and purpose-driven static relationship. We are not reusing the parameters of different model patches; we are merely using them as a starting point for training. Once again, a dynamic training process is required to adapt to drifting data. We believe that related tasks learn something in common for the same set of drift data and can use this to speed up the training process, whereas task relevance is static.

5.3 Entropy-based Sample Selection

We apply a standard entropy-based sample selection scheme in PatchLine. Specifically, we calculate the entropy of the inference results on a given sample as

$$I_{Entropy} = -\sum_{x \in X} p(x) \log p(x) \tag{5}$$

where $I_{Entropy}$ is the entropy of the inference results for samples, X is the inference results for samples, and x is the score of the inference results. This score represents the probability that the input data belongs to a certain category. For basic visual models such as image classification and object detection, input image, they produce a set of predicted class scores. The confidence of the model's predictions is reflected in the scores, and we typically choose the top1 score as the prediction output. When the model's predictions are inaccurate for an image, the entropy of the output scores will be low. Conversely, when the model is highly confident in its predictions, the output scores will have a distinct peak, resulting in high entropy. As the degree of data drift increases, the inference accuracy decreases, and the entropy of the output results rises [71], [72]. If $I_{Entropy} < I_{Source}$, we remove this sample from the training set. I_{Source} represents the average entropy of inference results of source data on the visual model, which can be obtained offline.

Note that it is common to capture an image as a training sample every few frames after detecting data drift [11]. Since most visual applications require continuous processing of video streams, such as autonomous driving, unmanned stores, and video surveillance, our method does not incur additional computational overhead.

6 Evaluation

6.1 Experimental Setups

Tasks and Models. We select three widely used vision tasks: image classification, object detection, and semantic segmentation. We construct nine DNN models using four backbones VGG16, ResNet50, MobileNetv2, and Swin-Transformer-Tiny, as shown in Table 2. VGG16 is a convolutional backbone network, consisting of 13 convolutional

TABLE 2
Overview of visual tasks and backbones (Swin-Tiny: Swin-Transformer-Tiny).

Backbone	Vision Task	DNN Model / Parameters (M)
VGG16	Image Classification	VGG16/134.29
	Object Detection	SSD-VGG16 /24.41
	Semantic Segmentation	U-Net-VGG16 / 24.89
ResNet50	Image Classification	ResNet50 / 23.52
	Object Detection	RetinaNet-ResNet50 /36.45
	Semantic Segmentation	U-Net-ResNet50 / 43.93
MobileNetv2	Image Classification	MobileNetv2 /2.23
	Object Detection	YOLOv4-MobileNetv2 / 12.10
	Semantic Segmentation	DeepLabv3+-MobileNetv2 / 5.82
Swin-Tiny	Image Classification	Swin-Tiny / 28.27
	Object Detection	RetinaNet-Swin-Tiny / 37.98
	Semantic Segmentation	DeepLabv3+-Swin-Tiny / 34.84

TABLE 3
Number of patches added on different datasets and amount of data after filtering (Swin-Tiny: Swin-Transformer-Tiny).

Backbone	Dataset	Patch	Entirety data/After filtering
VGG16	Core50-NC	2	144/120
	Core50-NI	3	144/130
	Core50-NIC	3	144/125
	SBD+	3	5469/5001
ResNet50	Core50-NC	3	144/126
	Core50-NI	4	144/135
	Core50-NIC	4	144/130
	SBD+	4	5469/4765
MobileNetv2	Core50-NC	4	144/128
	Core50-NI	7	144/132
	Core50-NIC	7	144/130
	SBD+	7	5469/5001
Swin-Tiny	Core50-NC	3	144/132
	SBD+	4	5469/5037

layers and 3 fully connected layers, with a total of 134.29 M parameters. Resnet50 is a residual backbone network, consisting of 4 stages and 50 layers, with a total of 23.52 M parameters. MobileNet2 is a lightweight backbone network, consisting of 7 bottleneck depth-separable convolution layers, with a total of 2.23 M parameters. Swin-Transformer-Tiny is a lightweight backbone network based on transformers, consisting of one patch embedding layer and 4 Swin transformer stages, with a total of 28.27 M parameters.

Datasets. Core50 [61] is a benchmark for model adaptation in three drift scenarios. It supports image classification, object detection and semantic segmentation. It contains 50 domestic objects from 10 categories and 11 distinct sessions (8 indoor and 3 outdoor). The detailed information about the dataset in the appendix.

We construct four datasets from Core50:

- Source Domain Dataset: It includes 1920 images of size 350x350 from 6 different categories,in the same scene conditions. The training and test sets contain 1800 and 120 images, respectively. This dataset is used for model pre-training and offline phase (Sec. 4).
- Core50-NC: It includes 180 images of size 350x350. The training and test sets contain 144 and 36 images, respectively.
- Core50-NI: It includes 174 images of size 350x350. The training and test sets contain 144 and 30 images, respectively.
- Core50-NIC: It includes 210 images of size 350x350. The training and test sets contain 144 and 66 images, respectively.

To experiment with larger datasets, we also create another benchmark called SBD+, from the SBD [73] dataset and the VOC2012 [74] dataset, which contains 11,391 images with 20 different object classes. Each image supports three visual tasks: image classification, object detection, and semantic segmentation. We construct two datasets from SBD+:

- Source Domain Dataset: We randomly select 40% of the data from SBD+ as source domain data (4556), then split this source domain dataset into 80% for the training set (3644) and 20% for the test set (912). The usage of the train and test sets, as well as the image dimensions, are consistent with the Core50 dataset.
- Drift Dataset: We randomly select 60% of the images (6835) from the SBD+ dataset and applied corruptions to images [13] to create the drift dataset. We split this source domain dataset into 80% for the training set (5469) and 20% for the test set (1366). The usage of the train and test sets, as well as the image dimensions, are consistent with the Core50 dataset.

Evaluation Platforms. We utilize two hardware platforms: the NVIDIA GTX 2080ti and the NVIDIA Jetson TX2 [22]. These two platforms have similar computing capabilities as representative edge devices. Specifically, the GTX 2080ti, with 11 GB RAM and 13.45 TFLOPS (FP32), serves as a representative for high-end edge devices, *e.g.*, the NVIDIA Jetson Orin NX [75], which has 16GB RAM and achieves 18.33 TFLOPS (FP32). Conversely, the NVIDIA Jetson TX2, with 8 GB RAM and 0.63 TFLOPS (FP32), represents entry-level edge devices. We also integrate the Jetson TX2 with an AgileX smart car [76] for a case study. Unless otherwise noted, our results are reported with the GTX 2080ti. Our PatchLine is implemented by Pytorch 1.7 and CUDA 10.0.

Baselines. We compare the performance of PatchLine with the following baselines. Training Details of PatchLine and baselines in the appendix.

- FT-full: Fine-tuning the full network.
- FT-last: Fine-tuning the last layer of the network [16].
- LST: The state-of-the-art model adaptation method [32]. It adds a small and separate network (known as ladder side network) to the backbone as an adapter and freezes other model parameters during training. For fair comparison, we have reproduced the ladder network structure in our three vision tasks according to the open-source implementation [77].

6.2 Overall Performance

6.2.1 Performance with Convolutional Backbones

In this thread of experiments, we evaluate our method on three convolutional backbones and four datasets.

Reduction in Epochs E_i. Table 4 shows the convergence epochs of different methods. Compared with FT-full and LST, we achieve the same accuracy, but reduce the number of epochs by an average of 56% and 50%, respectively. Note that FT-last has low accuracy on many tasks, because they require feature maps from intermediate layers. We also perform t-tests (see * and ** Table 5) to check whether the gains

TABLE 4
Accuracy and number of convergence epochs on convolutional backbones.

Dataset	Method	Image Classification Accuracy(%) ↑/Epochs ↓	Object Detection mAP(%)↑/Epochs↓	Semantic Segmentation mIoU(%)↑/Epochs↓
VGG16				
Core50-NC	FT-full	$100.00_{\pm 0.00}$/30	$100.00_{\pm 0.00}$/30	$90.13_{\pm 0.13}$/60
	FT-last	$95.32_{\pm 1.56}$/30	$59.77_{\pm 0.23}$/10	$52.72_{\pm 0.28}$/60
	LST	$95.32_{\pm 1.56}$/40	$100.00_{\pm 0.00}$/30	$89.00_{\pm 0.47}$/60
	PatchLine	**100.00**$_{\pm 0.00}$ **/20**	**99.60**$_{\pm 0.40}$ **/10**	**90.10**$_{\pm 0.35}$**/40**
Core50-NI	FT-full	$98.44_{\pm 1.56}$/30	$95.00_{\pm 1.67}$/40	$80.25_{\pm 0.44}$/60
	FT-last	$96.88_{\pm 3.13}$/80	$28.72_{\pm 0.17}$/10	$35.79_{\pm 0.42}$/60
	LST	$98.44_{\pm 1.56}$/40	$99.72_{\pm 0.23}$/30	$80.86_{\pm 0.10}$/60
	PatchLine	**98.44**$_{\pm 1.56}$ **/20**	**99.72**$_{\pm 0.28}$**/10**	**80.58**$_{\pm 0.04}$**/40**
Core50-NIC	FT-full	$99.22_{\pm 0.78}$/30	$97.73_{\pm 0.77}$/40	$84.89_{\pm 0.34}$/60
	FT-last	$95.32_{\pm 1.56}$/40	$38.56_{\pm 0.36}$/10	$35.19_{\pm 0.40}$/50
	LST	$97.66_{\pm 0.78}$/30	$99.72_{\pm 0.15}$/40	$84.16_{\pm 0.23}$/60
	PatchLine	**99.22**$_{\pm 0.78}$ **/20**	**99.43**$_{\pm 0.57}$**/10**	**84.30**$_{\pm 0.26}$**/40**
SBD+	FT-full	$85.66_{\pm 0.30}$/50	$74.63_{\pm 0.22}$/60	$55.68_{\pm 0.36}$/60
	FT-last	$64.00_{\pm 0.03}$/20	$16.53_{\pm 0.31}$/10	$13.63_{\pm 0.11}$/20
	LST	$82.04_{\pm 0.15}$/30	$74.06_{\pm 0.15}$/40	$55.38_{\pm 0.12}$/60
	PatchLine	**85.20**$_{\pm 0.14}$**/20**	**74.26**$_{\pm 0.22}$**/20**	**55.30**$_{\pm 0.16}$**/30**
ResNet50				
Core50-NC	FT-full	$100.00_{\pm 0.00}$/30	$100.00_{\pm 0.00}$/40	$90.80_{\pm 0.15}$/50
	FT-last	$95.32_{\pm 1.56}$/70	$83.27_{\pm 0.29}$/50	$47.99_{\pm 0.72}$/30
	LST	$95.32_{\pm 1.56}$/70	$98.61_{\pm 1.39}$/50	$90.30_{\pm 0.65}$/50
	PatchLine	**100.00**$_{\pm 0.00}$**/20**	**100.00**$_{\pm 0.00}$**/10**	**90.68**$_{\pm 0.26}$**/20**
Core50-NI	FT-full	$98.44_{\pm 1.56}$/40	$93.05_{\pm 0.83}$/50	$79.72_{\pm 0.23}$/60
	FT-last	$96.88_{\pm 3.12}$/80	$75.65_{\pm 0.33}$/60	$29.90_{\pm 0.20}$/40
	LST	$96.88_{\pm 3.12}$/50	$95.56_{\pm 0.56}$/40	$79.45_{\pm 0.15}$/60
	PatchLine	**100.00**$_{\pm 0.00}$**/20**	**99.72**$_{\pm 0.28}$**/10**	**79.69**$_{\pm 0.21}$**/30**
Core50-NIC	FT-full	$100.00_{\pm 0.00}$/40	$96.65_{\pm 0.19}$/50	$84.36_{\pm 0.14}$/60
	FT-last	$92.97_{\pm 0.79}$/50	$74.02_{\pm 0.08}$/50	$30.39_{\pm 0.40}$/30
	LST	$98.44_{\pm 1.56}$/50	$98.39_{\pm 0.16}$/40	$84.64_{\pm 0.17}$/60
	PatchLine	**99.22**$_{\pm 0.78}$**/20**	**99.56**$_{\pm 0.44}$**/20**	**84.60**$_{\pm 0.09}$**/30**
SBD+	FT-full	$85.77_{\pm 0.23}$/50	$77.04_{\pm 0.17}$/50	$61.73_{\pm 0.26}$/60
	FT-last	$76.98_{\pm 0.22}$/70	$17.30_{\pm 0.23}$/40	$10.40_{\pm 0.21}$/40
	LST	$84.34_{\pm 0.18}$/40	$76.89_{\pm 0.15}$/40	$61.36_{\pm 0.31}$/60
	PatchLine	**84.90**$_{\pm 0.11}$**/20**	**76.94**$_{\pm 0.06}$**/20**	**61.64**$_{\pm 0.06}$**/40**
MobileNetv2				
Core50-NC	FT-full	$98.44_{\pm 1.56}$/50	$88.83_{\pm 0.33}$/60	$91.08_{\pm 0.09}$/60
	FT-last	$95.31_{\pm 1.56}$/30	$49.26_{\pm 0.25}$/10	$46.58_{\pm 0.40}$/50
	LST	$96.88_{\pm 0.00}$/20	$87.44_{\pm 0.20}$/50	$89.08_{\pm 0.40}$/40
	PatchLine	**98.44**$_{\pm 1.56}$**/10**	**88.20**$_{\pm 0.16}$**/40**	**90.41**$_{\pm 0.06}$**/20**
Core50-NI	FT-full	$100.00_{\pm 0.00}$/20	$88.23_{\pm 0.18}$/50	$89.41_{\pm 0.39}$/50
	FT-last	$90.63_{\pm 3.13}$/60	$67.66_{\pm 0.21}$/20	$38.27_{\pm 0.26}$/50
	LST	$98.44_{\pm 1.56}$/20	$87.44_{\pm 0.10}$/40	$87.36_{\pm 0.30}$/40
	PatchLine	**100.00**$_{\pm 0.00}$**/10**	**87.40**$_{\pm 0.12}$**/20**	**88.43**$_{\pm 0.23}$**/20**
Core50-NIC	FT-full	$99.22_{\pm 0.78}$/60	$89.94_{\pm 0.02}$/60	$89.66_{\pm 0.27}$/60
	FT-last	$92.97_{\pm 0.79}$/70	$54.19_{\pm 0.12}$/10	$38.41_{\pm 0.08}$/40
	LST	$96.10_{\pm 0.70}$/40	$88.08_{\pm 0.31}$/50	$88.23_{\pm 0.22}$/40
	PatchLine	**99.22**$_{\pm 0.78}$**/10**	**88.75**$_{\pm 0.04}$**/30**	**89.17**$_{\pm 0.13}$**/20**
SBD+	FT-full	$82.94_{\pm 0.22}$/60	$70.16_{\pm 0.17}$/60	$64.01_{\pm 0.10}$/70
	FT-last	$75.75_{\pm 0.27}$/70	$10.63_{\pm 0.26}$/20	$16.11_{\pm 0.12}$/60
	LST	$81.89_{\pm 0.11}$/40	$69.45_{\pm 0.14}$/40	$63.44_{\pm 0.10}$/60
	PatchLine	**82.30**$_{\pm 0.07}$**/10**	**69.97**$_{\pm 0.02}$**/20**	**63.64**$_{\pm 0.36}$**/30**

are statistically significant. Overall, PatchLine achieves high accuracy on various drift datasets with fewer convergence epochs across the three convolutional backbones.

Reduction in Single-pass Computational Cost $(F_i + B_i)$**.** Fig. 8 shows the corresponding single-pass computational cost of different methods. PatchLine reduces the single-pass computational cost by up to an average of 64% than FT-full. Compared with LST, our single-pass computational cost is lower under various drift datasets. For example, with VGG16 backbone on Core50-NC dataset, we only need two patches, reducing the single-pass computational cost by 5%. Although our single-pass computational cost is slightly higher than FT-last, we achieve much higher accuracy.

Reduction in Training Samples n_i**.** As shown in Table 3, our entropy-based sample selection reduces the amount of training data by an average of 11% on three backbones.

Overall Reduction in Computation and Latency. As shown in Fig. 8, on three backbones and four datasets , we reduce the total computational cost by an average of 85% compared to FT-full. Compared to LST, we achieve higher accuracy and reduce the total computational cost by an average of 55%. In comparison to FT-full and LST, we respectively decrease the total training time by 70% and 51%, as shown in Fig. 8. Especially, using ResNet50 as the backbone on the Core50-NC dataset, we update the three vision tasks in just 100 seconds. In summary, PatchLine consistently achieves faster multi-DNN model fine-tuning with less computational overhead across the three backbones.

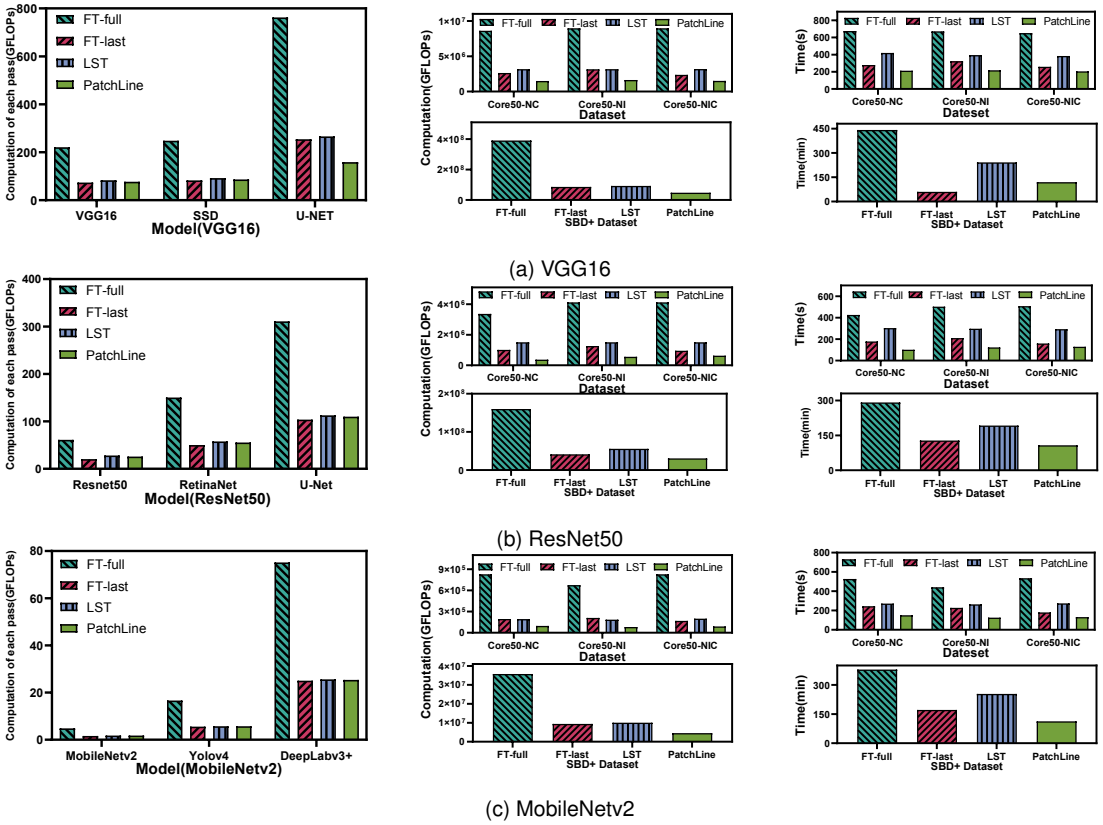


Fig. 8. Single-pass computational cost, total computational cost and training time with different convolutional backbones: (a) VGG16; (b) ResNet50; (c) MobileNetv2.

TABLE 5
Paired t-test results and p-values on four backbones and four datasets. *: p-value <0.05; **: p-value <0.01.IC:Image Classification, OD: Object Detection, SS: Semantic Segmentation.

Dataset	IC p-value	OD p-value	SS p-value
VGG16			
Core50-NC	$5.99 \times 10^{-11**}$	$2.12 \times 10^{-2*}$	$1.57 \times 10^{-16**}$
Core50-NI	$1.33 \times 10^{-2*}$	$1.57 \times 10^{-2*}$	$6.94 \times 10^{-17**}$
Core50-NIC	$0.06 \times 10^{-2**}$	$2.57 \times 10^{-5**}$	$1.39 \times 10^{-2*}$
SBD+	$1.57 \times 10^{-42**}$	$0.10 \times 10^{-2**}$	$1.30 \times 10^{-2*}$
ResNet50			
Core50-NC	$1.38 \times 10^{-10**}$	$3.49 \times 10^{-6**}$	$7.29 \times 10^{-5**}$
Core50-NI	$2.10 \times 10^{-2*}$	$0.21 \times 10^{-2**}$	$3.73 \times 10^{-9**}$
Core50-NIC	$3.36 \times 10^{-2*}$	$6.31 \times 10^{-13**}$	$1.16 \times 10^{-2*}$
SBD+	$2.73 \times 10^{-12**}$	$6.37 \times 10^{-17**}$	$2.52 \times 10^{-7**}$
MobileNetv2			
Core50-NC	$5.99 \times 10^{-11**}$	$1.84 \times 10^{-21**}$	$8.19 \times 10^{-17**}$
Core50-NI	$8.07 \times 10^{-14**}$	$0.02 \times 10^{-2**}$	$1.16 \times 10^{-22**}$
Core50-NIC	$2.12 \times 10^{-23**}$	$2.45 \times 10^{-13**}$	$1.17 \times 10^{-24**}$
SBD+	$2.18 \times 10^{-15**}$	$1.63 \times 10^{-17**}$	$1.27 \times 10^{-5**}$
Swin-Tiny			
Core50-NC	$4.25 \times 10^{-2*}$	$9.43 \times 10^{-9**}$	$2.99 \times 10^{-6**}$
SBD+	$2.33 \times 10^{-12**}$	$1.22 \times 10^{-9**}$	$6.24 \times 10^{-15**}$

Peak Memory Cost. Our PatchLine is also memory-efficient, since the patches are lightweight and only the patch parameters are updated during backpropagation. As shown in Fig. 9, we reduce the peak memory cost by up to 44% and 8% compared to FT-full and LST, respectively. Although Ft-last achieves the lowest peak memory cost, updating only the last layer fails to meet the accuracy requirements.

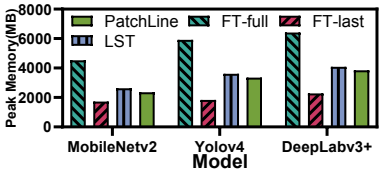


Fig. 9. Peak memory of different methods (MobileNetv2 backbone).

6.2.2 *Performance with Transformer-Based Backbone*

In this thread of experiments, we use Swin-Transformer-Tiny as the backbone. Swin-Tiny consists of four Swin transformer stages, and we add patches to each stage using residual connections.

Reduction in Epochs E_i. As shown in Table 6, across the two datasets, we reduce the number of epochs by 57% and 44% compared with Ft-full and LST, respectively.

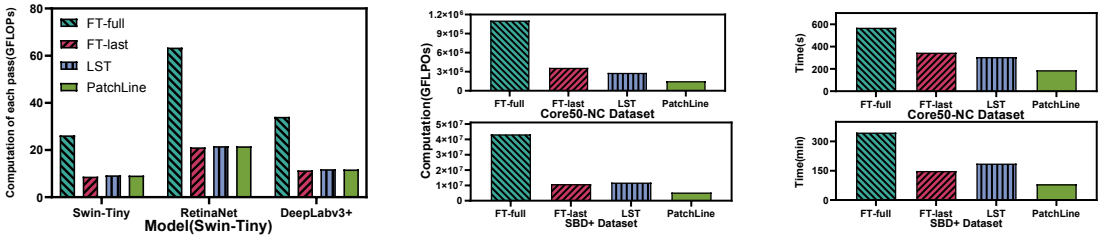
Reduction in Single-pass Computational Cost $(F_i + B_i)$. As shown in Fig. 10 (a), PatchLine reduces the single-pass computational cost by up to 65% compared with Ft-full.

Reduction in Training Samples n_i. As shown in Table 3, our sample selection method reduces the amount of training data by 8%.

Overall Reduction in Computation and Latency. As shown in Fig. 10 (b), we reduce the total computational cost by up to 86% and 50% compared to FT-full and LST. In comparison to FT-full and LST, we respectively diminish the total train-

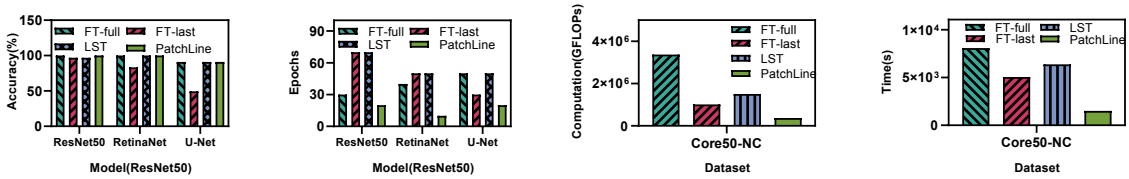
TABLE 6
Accuracy and number of convergence epochs with transformer-based backbone.

Dataset	Method	Image Classification Swin-Tiny Accuracy(%) ↑/Epochs ↓	Object Detection RetinaNet-Swin-Tiny mAP(%)↑/Epochs↓	Semantic Segmentation DeepLabv3+-Swin-Tiny mIoU(%)↑/Epochs↓
Core50-NC	FT-full	$100.00_{\pm 0.00}$/30	$74.69_{\pm 0.28}$/70	$69.69_{\pm 0.26}$/60
	FT-last	$96.88_{\pm 1.56}$/40	$50.55_{\pm 0.43}$/70	$47.55_{\pm 0.34}$/60
	LST	$98.44_{\pm 1.56}$/30	$74.14_{\pm 0.22}$/50	$68.76_{\pm 0.75}$/50
	PatchLine	$\mathbf{100.00}_{\pm 0.00}$/**20**	$\mathbf{74.44}_{\pm 0.12}$/**30**	$\mathbf{69.44}_{\pm 0.11}$/**30**
SBD+	FT-full	$88.70_{\pm 0.22}$/40	$47.66_{\pm 0.23}$/70	$52.91_{\pm 0.16}$/70
	FT-last	$80.62_{\pm 0.18}$/30	$7.15_{\pm 0.11}$/60	$12.49_{\pm 0.19}$/40
	LST	$87.66_{\pm 0.29}$/30	$47.20_{\pm 0.26}$/60	$52.27_{\pm 0.23}$/50
	PatchLine	$\mathbf{88.08}_{\pm 0.14}$/**20**	$\mathbf{47.60}_{\pm 0.06}$/**30**	$\mathbf{52.80}_{\pm 0.08}$/**20**



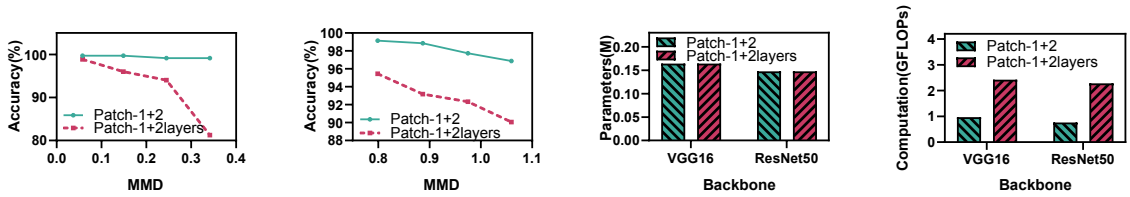
(a) Single-pass computational cost (b) Total computational cost (c) Total training time

Fig. 10. Single-pass computational cost, total computational cost and training time with transformer-based backbone.



(a) Accuracy (b) Convergence epochs (c) Total computational cost (d) Total training time

Fig. 11. Accuracy, number of convergence epochs, total computational cost and training time on Jetson TX2 (ResNet50 backbone).



(a) Accuracy-VGG16 (b) Accuracy-ResNet50 (c) Parameters (d) Computation

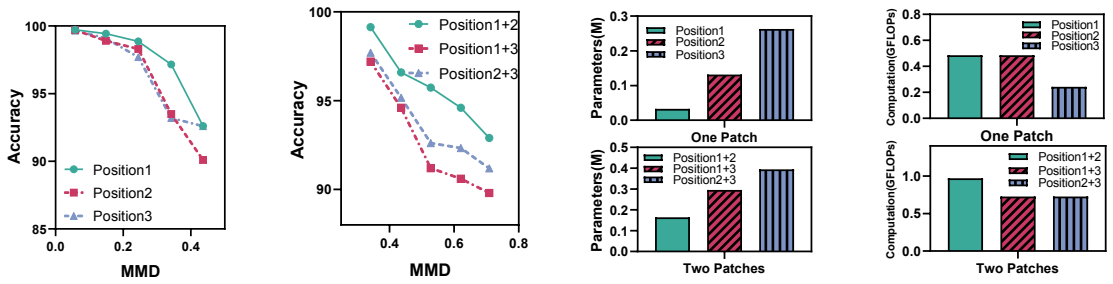
Fig. 12. Performance of two extension methods with different backbones. Patch-1+2: already patch-1, adding a patch to the 2-th convolution group within model. Patch-1+2layers: already patch-1, adding two convolution layers within the patch-1.

ing time by 71% and 47%, as shown in Fig. 10 (c). Overall, the experimental results demonstrate that our method can also achieve efficient fine-tuning of multiple DNN models on the transformer-based backbone.

6.2.3 *Performance on Jetson TX2*

In this thread of experiments, we evaluate the performances of PatchLine and baselines on the Jetson TX2.

Reduction in epochs E_i. The PatchLine and baselines show consistent accuracy and convergence epochs on the TX2 platform. As shown in Fig. 11 (a-b), compared with FT-full and LST, we achieve the same accuracy, but reduce the number of epochs by 58% and 71%, respectively. **Overall Reduction in Computation and Latency.** Due to the limited AI computing capabilities of the Jetson TX2, PatchLine requires 15× training time compared to deployment on the GTX 2080 Ti. However, compared to the baselines, we still significantly reduce computational cost and training time. From Fig. 11 (c), we reduce the total computational cost by up to 88% and 75% compared to FT-full and LST, respectively. Compared with FT-full and LST, we decrease the total training time by 81% and 75%, as in Fig. 11 (d).



(a) Accuracy-one patch (b) Accuracy-two patches (c) Parameters (d) Computation

Fig. 13. Performance of adding patch in order and skipping the convolution group. Position i means adding a patch to the i-th convolution group. Position i+j means adding patches to the i-th and j-th convolution group.

6.3 Comparison of Patch Extension Methods

6.3.1 Expansion within Model vs. within Patch

We assess the effectiveness of two distinct patch extension methods in image classification with two backbones. Specifically, we compare their performance in terms of accuracy, parameters, and computational cost under various drift levels (measured by MMD). From Fig. 12, we observe that adding patches within the model and maintaining the same parameter count, leads to an average accuracy improvement of 7% compared to the alternative. This method also reduces the computational cost by up to 63% than adding convolution layers inside the patch.

6.3.2 Continual Patching vs. Skip Patching

This experiment evaluates the performance of two patching methods: continual patching and skip patching, on image classification task and with the VGG16 backbone. From Fig. 13 (a-b), we observe that regardless of whether one or two patches are added, placing patches in the sequence of convolution groups consistently yielded high accuracy. Interestingly, continual patching requires fewer parameters compared to skip patching. This is further shown in Fig. 13 (c), where adding two patches at positions 1 and 2, as opposed to positions 1 and 3, results in a 58% reduction in parameters. This strategy also improves the accuracy by up to 3% (from 92.61% to 95.74%) under an MMD of 0.5280. In contrast, the computational cost of continual patching is marginally higher than that of skip patching. For instance, the workload at positions 1+2 is only 0.2424 GFLOPs greater than at positions 1+3.

6.4 Ablation Studies

6.4.1 Effectiveness of Entropy-based Sample Selection

The purpose of this experiment is to examine whether our entropy-based sample selection affects accuracy and convergence time. As shown by Fig. 14, our sample selection method does not affect the accuracy of the results, but rather reduces the training time. Taking the example (Fig. 14 (a)) of image classification task with VGG16 as the backbone and the dataset being core50-NC, adding the sample selection method can reduce training time by 17% (from 26.62s to 22.18s) while achieving the same accuracy. In summary, the results of two distinct data sets indicate that the method of sample selection reduces the training time overhead by an average of 13% and 9% on VGG16 and ResNet50, respectively, without compromising accuracy.

TABLE 7
Computational cost introduced by adding different numbers of patches. Patch-x: adding x patches.

Backbone	Patch-x	Computation (GFLOPs)
MobileNetv2	Patch-7	0.065
	Patch-4	0.048
	Patch-2	0.043
ResNet50	Patch-4	1.776
	Patch-3	1.269
	Patch-1	0.254

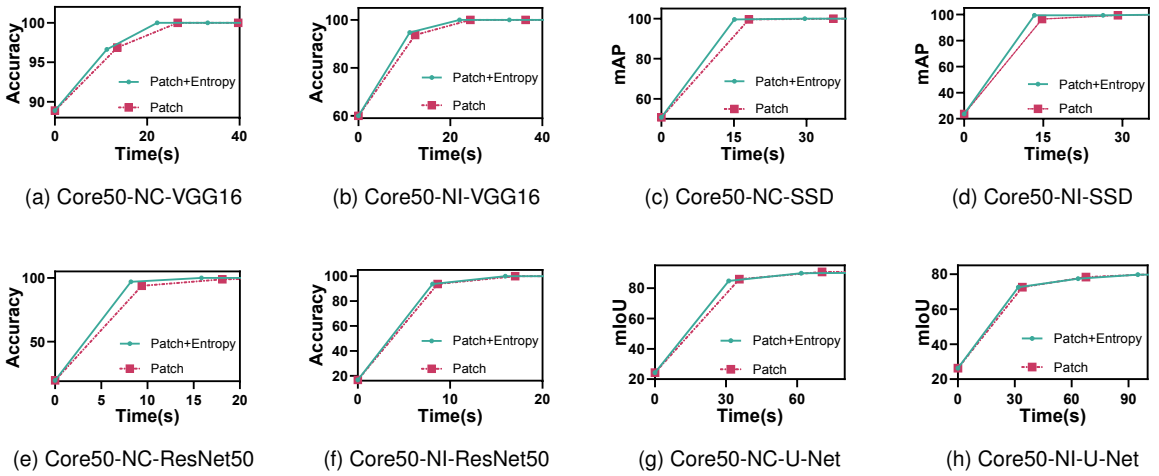
The reasons are as follows. *(i)* Filtering data similar to the source does not affect accuracy, as we freeze the model trained on the source data, preserving the knowledge from the source data. *(ii)* The training data is derived from a video stream, where redundancy exists between frames. Thus, sampling training data does not substantially decrease the overall information. Some studies [13], [80] suggest that reducing the training samples can even improve accuracy.

6.4.2 Effectiveness of Warm Patching

In this section, we compare the accuracy and convergence time of our warm patching method with three common random initialization methods (Normal [78], Xavier [79], Uniform [78]). From Fig. 15, our warm patching method consistently achieves higher accuracy and shorter convergence times. The other initialization methods yield similar accuracy, yet demand more training time. For instance, on the U-Net-ResNet50 model, our approach reduces training time by an average of 54% compared to other methods on the two datasets. For the eight scenarios in Fig. 15, our method reduces the training time by up to 55% than other methods. In some cases, such as the U-Net-VGG16 model, our method improves the accuracy by 5%-3% compared to other methods. In summary, our warm patching method outperforms various random initialization approaches, achieving higher training accuracy and faster convergence.

6.4.3 Effectiveness of Patch Predictor

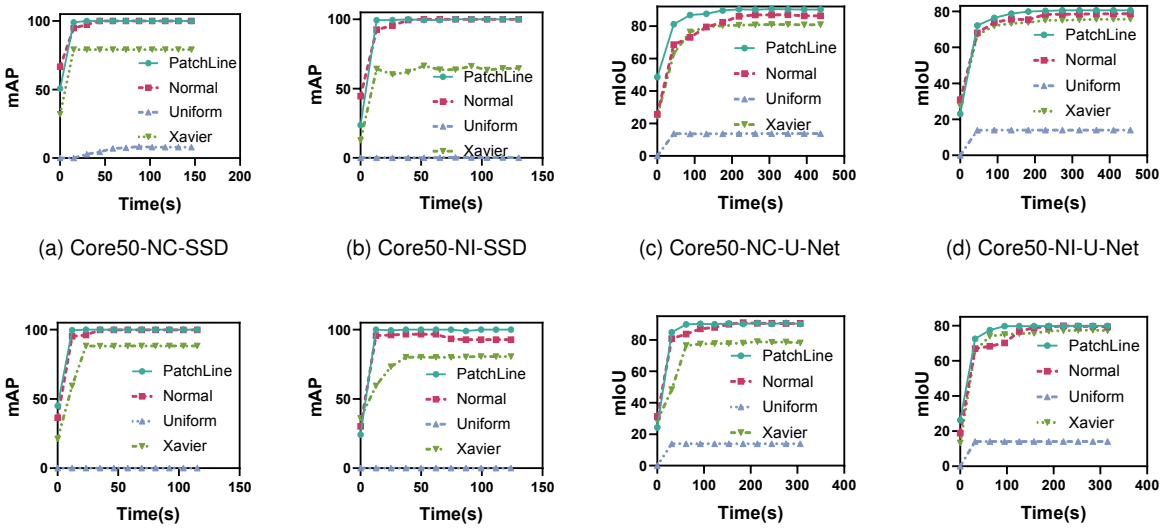
In this experiment, we examine whether the patch predictor has an impact on accuracy and computation cost. The patch



(a) Core50-NC-VGG16 (b) Core50-NI-VGG16 (c) Core50-NC-SSD (d) Core50-NI-SSD

(e) Core50-NC-ResNet50 (f) Core50-NI-ResNet50 (g) Core50-NC-U-Net (h) Core50-NI-U-Net

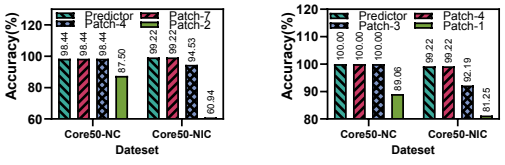
Fig. 14. Impact of entropy-based sample selection. Patch: adding patches and utilizing the entirety of the training samples. Patch+Entropy: adding patches and filtering training samples based entropy. VGG16: a-b and ResNet50: e-h.



(a) Core50-NC-SSD (b) Core50-NI-SSD (c) Core50-NC-U-Net (d) Core50-NI-U-Net

(e) Core50-NC-RetinaNet (f) Core50-NI-RetinaNet (g) Core50-NC-U-Net (h) Core50-NI-U-Net

Fig. 15. Performance comparison of our warm start for patches and three common random initialization methods (Normal [78], Xavier [79] and Uniform [78]). VGG16: a-d and ResNet50: e-h.



(a) MobileNetv2 (b) ResNet50

Fig. 16. Impact of patch predictor on accuracy. Predictor: adding patches based on the predictor's results. Patch-x: adding x patches.

predictor selects the optimal number of patches based on the degree of data drift, reducing computational overhead without compromising accuracy. As shown in Table 3, the patch predictor consistently selects fewer patches on the Core-NC dataset compared to the Core-NIC dataset, adding 4 and 3 patches for the MobileNetv2 and ResNet50 backbones, respectively. This is because dataset Core-NC has a smaller degree of drift compared to dataset Core-NIC. As shown in Fig. 16, on the Core-NC dataset, the patch predictor achieved the same accuracy as adding the maximum number of patches (Patch-7 in MobileNetv2 and Patch-4 in ResNet50) on both backbone networks. As shown in Table 7, by reducing the number of patches, the patch predictor reduced the computational overhead by 26% and 29% compared to adding the maximum number of patches on both backbones, respectively. However, insufficient patch additions can have a significant impact on accuracy. On the MobileNetV2 backbone network, the Patch Predictor improved accuracy by 8% and 62% on the Core-NIC dataset compared to Patch-3 and Patch-1, respectively. In summary,

TABLE 8
Accuracy and number of convergence epochs with ResNet50 backbone on the autonomous driving application.

Condition	Method	Image Classification ResNet50 Accuracy(%)↑/Epochs↓	Object Detection RetinaNet-ResNet50 mAP(%)↑/Epochs↓	Semantic Segmentation U-Net-ResNet50 mIoU(%)↑/Epochs↓
Snow	FT-full	100.00/40	60.81/50	57.67/70
	FT-last	100.00/40	18.33/30	16.67/40
	LST	100.00/60	59.92/50	56.12/70
	PatchLine	**100.00/20**	**61.33/20**	**57.89/40**
Rain	FT-full	100.00/30	69.06/40	81.98/60
	FT-last	100.00/50	30.70/40	20.07/10
	LST	100.00/40	69.22/40	77.93/60
	PatchLine	**100.00/20**	**69.23/20**	**82.18/30**
Night	FT-full	93.75/30	50.89/70	44.80/60
	FT-last	81.25/40	17.43/50	7.21/40
	LST	81.25/40	48.52/60	43.96/50
	PatchLine	**93.75/20**	**49.46/30**	**43.78/30**

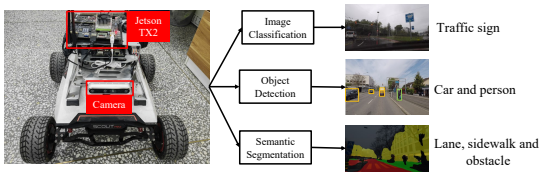
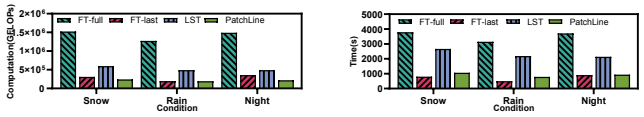


Fig. 17. An illustration of case study, where we deploy PatchLine and baselines on a smart car as an autonomous driving application.



(a) Total computational cost (b) Total training time

Fig. 18. Total computational cost and training time with ResNet50 backbone on the autonomous driving application.

our patch predictor can be applied to a variety of backbone networks and datasets, reducing computational overhead without compromising accuracy.

6.5 Case Study

We deploy PatchLine and various baselines on a smart car to update three vision tasks for an autonomous driving application. As depicted in Fig. 17, we utilize image classification to identify road signs, object detection for recognizing cars and people, and semantic segmentation to delineate lane boundaries. We construct datasets for evaluations based on the ACDC dataset [81], which includes images of car driving in diverse weather conditions like rain, snow, fog, and night.

- Source Domain Dataset: It includes 500 images of size 1920x1080 from fog condition. The training and test sets contain 400 and 100 images, respectively.
- Snow, Rain, and Night Dataset: Each condition includes 60 images of size 1920x1080. The training and test sets contain 50 and 10 images, respectively.

Due to the limited resource of the TX2 platform, we down-scale all the training samples to 350x350 pixels. We use ResNet50 as the backbone, and add 3, 4, and 4 patches respectively to handle drifts caused by rain, snow and night conditions. The training sequence is as follows: ResNet50, RetinaNet-ResNet50, and U-Net-ResNet50, with the latter two models' backbones being initialized using the patch parameters of ResNet50. Batch sizes are 8 for image classification, 8 for object detection, and 4 for semantic segmentation. The experiment results are shown below.

Reduction in Epochs (E_i). As shown in Table 8, our approach not only achieves higher accuracy compared to FT-full and LST but also reduces the number of training epochs by 57% and 51%, respectively. Specifically, for semantic segmentation under the rain condition, we observe a 5% accuracy improvement over LST. However, in the snow condition, the accuracy for lane detection is lower, likely due to snow coverage. At the night condition, due to insufficient lighting, the accuracy of all three visual tasks decreases.

Reduction in Training Samples (n_i). Our entropy-based sample selection strategy effectively reduces the amount of training data by 16%, 8%, and 8% for the rain, snow, and night condition.

Overall Reduction in Computation and Latency. From Fig. 18 (a), PatchLine reduces the total computational cost by up to 84% and 57% compared to FT-full and LST, respectively. Furthermore, from Fig. 18 (b), the overall training time decreases by 74% and 60% compared to FT-full and LST. In summary, PatchLine not only demonstrates superior accuracy across diverse weather conditions but also substantially reduces computational costs.

7 LIMITATIONS

As a first step towards efficient multi-DNN fine-tuning, our work still has some limitations. *(i)* The model training sequence is obtained offline given prior knowledge of the vision tasks. Accordingly, when there is a change in the targeted vision tasks, the training sequence needs to be updated. The profiling overhead may limit our method to applications with fixed vision tasks. *(ii)* Our method relies on data labels. In many real-world mobile computing applications, labels are not always readily available. Therefore, multi-DNN adaptation in a semi-supervised or unsupervised manner remains a challenge. *(iii)* Our solution is evaluated with only three vision tasks. It would be beneficial

to test our method on other visual tasks and even extend it to other application domains.

8 Future Work

In the future, we aspire to enhance PatchLine in the following areas: *(i)* The current design of PatchLine is built upon supervised training (fine-tuning). Given the increasing interest in unsupervised domain adaptation [82], we plan to extend the principles of PatchLine to unsupervised on-device multi-DNN fine-tuning. *(ii)* Our primary goal is to reduce the overall computation workload in training (fine-tuning), which often results in shorter overall training latency (see Sec. 6.2.1). Our PatchLine is also memory-efficient, since the patches are lightweight, and only lead to an average 56% peak memory (see Sec. 6.2.1) than updating all parameters during backpropagation. We plan to explore patch designs that are both computation- and memory-efficient in the future. *(iii)* PatchLine currently shares patch parameters only at the model level. Finer-grained parameter sharing *e.g.*, at the patch level would further reduce training latency.

9 Conclusion

We propose PatchLine, an efficient on-device model fine-tuning framework for multi-DNN vision applications. The core idea of PatchLine is to add a lightweight residual adaptation module (referred to as a patch) to the model and strategically update the patch module using the relevance of vision tasks. Specifically, PatchLine introduce a series techniques such as drift-adaptive incremental patching, correlation-aware warm patching, and entropy-based sample selection, to holistically reduce the number of trainable parameters, training epochs, and training samples. Extensive experiments on diverse datasets, vision tasks, backbones and platforms demonstrate PatchLine reduces the total computational cost by an average of 55% without sacrificing accuracy compared to the state-of-the-art model adaptation framework.

References

[1] D. Roy, Y. Li, T. Jian, P. Tian, K. Chowdhury, and S. Ioannidis, "Multi-modality sensing and data fusion for multi-vehicle detection," *IEEE Transactions on Multimedia*, vol. 25, pp. 2280–2295, 2023.

[2] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 68–81.

[3] J. Wang, G. Wang, X. Zhang, L. Liu, H. Zeng, L. Xiao, Z. Cao, L. Gu, and T. Li, "Patch: A plug-in framework of non-blocking inference for distributed multimodal system," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 7, no. 3, pp. 1–24, 2023.

[4] K. Qu, W. Zhuang, W. Wu, M. Li, X. Shen, X. Li, and W. Shi, "Stochastic cumulative dnn inference with rl-aided adaptive iot device-edge collaboration," *IEEE Internet of Things Journal*, 2023.

[5] Z. Huang, S. Yang, M. Zhou, Z. Gong, A. Abusorrah, C. Lin, and Z. Huang, "Making accurate object detection at the edge: Review and new approach," *Artificial Intelligence Review*, vol. 55, no. 3, pp. 2245–2274, 2022.

[6] J. Yao, Y. Li, C. Liu, and R. Tang, "Ehsinet: Efficient high-order spatial interaction multi-task network for adaptive autonomous driving perception," *Neural Processing Letters*, pp. 1–18, 2023.

[7] P. Tu, H. Ye, J. Young, M. Xie, C. Zheng, and X. Chen, "Efficient spatiotemporal learning of microscopic video for augmented reality-guided phacoemulsification cataract surgery," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2023, pp. 682–692.

[8] J. Ahn, J. Park, S. S. Lee, K.-H. Lee, H. Do, and J. Ko, "Safefac: Video-based smart safety monitoring for preventing industrial work accidents," *Expert Systems with Applications*, vol. 215, p. 119397, 2023.

[9] Y. Xie, F. Li, and Y. Wang, "Fingerslid: Towards finger-sliding continuous authentication on smart devices via vibration," *IEEE Transactions on Mobile Computing*, 2023.

[10] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Networks*, vol. 116, pp. 56–73, 2019.

[11] M. Khani, P. Hamadanian, A. Nasr-Esfahany, and M. Alizadeh, "Real-time video inference on edge devices via adaptive model streaming," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4572–4582.

[12] X. Zhou, Y. Tian, and X. Wang, "Mec-da: Memory-efficient collaborative domain adaptation for mobile edge devices," *IEEE Transactions on Mobile Computing*, 2023.

[13] Y. Gan, M. Pan, R. Zhang, Z. Ling, L. Zhao, J. Liu, and S. Zhang, "Cloud-device collaborative adaptation to continual changing environments in the real-world," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 12 157–12 166.

[14] H. Wen, Y. Li, Z. Zhang, S. Jiang, X. Ye, Y. Ouyang, Y. Zhang, and Y. Liu, "Adaptivenet: Post-deployment neural architecture adaptation for diverse edge environments," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–17.

[15] Y. Kong, P. Yang, and Y. Cheng, "Edge-assisted on-device model update for video analytics in adverse environments," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 9051–9060.

[16] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*. PMLR, 2014, pp. 647–655.

[17] C. Gan, N. Wang, Y. Yang, D.-Y. Yeung, and A. G. Hauptmann, "Devnet: A deep event network for multimedia event detection and evidence recounting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2568–2577.

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[19] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[20] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[21] H. Jiang, J. Starkman, Y.-J. Lee, H. Chen, X. Qian, and M.-C. Huang, "Distributed deep learning optimized system over the cloud and smart phone devices," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 147–161, 2019.

[22] NVIDIA, "Nvidia jetson tx2," Product Introduction, Oct. 2023. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.

[23] Z. Qu, Z. Zhou, Y. Tong, and L. Thiele, "p-meta: Towards on-device deep model adaptation," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1441–1451.

[24] H. Li, G. Hu, J. Li, and M. Zhou, "Intelligent fault diagnosis for large-scale rotating machines using binarized deep neural networks and random forests," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 2, pp. 1109–1119, 2021.

[25] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," *Advances in neural information processing systems*, vol. 30, 2017.

[26] J. Wei, X. Zhang, Z. Zhuo, Z. Ji, Z. Wei, J. Li, and Q. Li, "Leader population learning rate schedule," *Information Sciences*, vol. 623, pp. 455–468, 2023.

[27] F. Lai, Y. Dai, H. V. Madhyastha, and M. Chowdhury, "{ModelKeeper}: Accelerating {DNN} training via automated training warmup," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 769–785.

[28] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 285–11 297, 2020.

[29] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[30] W. Zhuang, Y. Wen, L. Lyu, and S. Zhang, "Mas: Towards resource-efficient federated multiple-task learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 23 414–23 424.

[31] A. M. L. Research, "A multi-task neural architecture for on-device scene analysis," Research, Jun. 2022. [Online]. Available: https://machinelearning.apple.com/research/on-device-scene-analysis

[32] Y.-L. Sung, J. Cho, and M. Bansal, "Lst: Ladder side-tuning for parameter and memory efficient transfer learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 991–13 005, 2022.

[33] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.

[34] R. T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian, "Online model distillation for efficient video inference," in *Proceedings of the IEEE/CVF International conference on computer vision*, 2019, pp. 3573–3582.

[35] G. Yin, J. Zhang, G. Shen, and Y. Chen, "Fewsense, towards a scalable and cross-domain wi-fi sensing system using few-shot learning," *IEEE Transactions on Mobile Computing*, 2022.

[36] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 119–135.

[37] L. Xu, X. Ding, H. Peng, D. Zhao, and X. Li, "Adtcd: An adaptive anomaly detection approach towards concept-drift in iot," *IEEE Internet of Things Journal*, 2023.

[38] J. Chauhan, Y. D. Kwon, P. Hui, and C. Mascolo, "Contauth: Continual learning framework for behavioral-based user authentication," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 4, pp. 1–23, 2020.

[39] S. Lee, B. Islam, Y. Luo, and S. Nirjon, "Intermittent learning: On-device machine learning on intermittently powered system," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 4, pp. 1–30, 2019.

[40] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.

[41] A. Mathur, D. J. Beutel, P. P. B. de Gusmao, J. Fernandez-Marques, T. Topal, X. Qiu, T. Parcollet, Y. Gao, and N. D. Lane, "On-device federated learning with flower," *arXiv preprint arXiv:2104.03042*, 2021.

[42] K. Huang, B. Yang, and W. Gao, "Elastictrainer: Speeding up on-device training with runtime elastic tensor selection," in *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, 2023, pp. 56–69.

[43] X. Zeng, M. Yan, and M. Zhang, "Mercury: Efficient on-device distributed dnn training via stochastic importance sampling," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 29–41.

[44] H. Gao, J. Feng, Y. Xiao, B. Zhang, and W. Wang, "A uav-assisted multi-task allocation method for mobile crowd sensing," *IEEE Transactions on Mobile Computing*, 2022.

[45] Y. Zhang, B. Guo, J. Liu, T. Guo, Y. Ouyang, and Z. Yu, "Which app is going to die? a framework for app survival prediction with multitask learning," *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 728–739, 2020.

[46] L. Chen, Y. Zhang, S. Miao, S. Zhu, R. Hu, L. Peng, and M. Lv, "Salience: An unsupervised user adaptation model for multiple wearable sensors based human activity recognition," *IEEE Transactions on Mobile Computing*, 2022.

[47] J. Lu, C. Shang, C. Yue, R. Morillo, S. Ware, J. Kamath, A. Bamis, A. Russell, B. Wang, and J. Bi, "Joint modeling of heterogeneous sensing data for depression assessment via multi-task learning," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, pp. 1–21, 2018.

[48] W. Gu, Y. Zhou, Z. Zhou, X. Liu, H. Zou, P. Zhang, C. J. Spanos, and L. Zhang, "Sugarmate: Non-intrusive blood glucose monitoring with smartphones," *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, vol. 1, no. 3, pp. 1–27, 2017.

[49] H. Ma, Z. Zhang, W. Li, and S. Lu, "Unsupervised human activity representation learning with multi-task deep clustering," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 1, pp. 1–25, 2021.

[50] R. Dai, T. Kannampallil, J. Zhang, N. Lv, J. Ma, and C. Lu, "Multi-task learning for randomized controlled trials: a case study on predicting depression with wearable data," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1–23, 2022.

[51] X. He, X. Wang, Z. Zhou, J. Wu, Z. Yang, and L. Thiele, "On-device deep multi-task inference via multi-task zipping," *IEEE Transactions on Mobile Computing*, 2021.

[52] X. He, D. Gao, Z. Zhou, Y. Tong, and L. Thiele, "Pruning-aware merging for efficient multitask inference," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 585–595.

[53] Z. Wang, X. He, Z. Zhou, X. Wang, Q. Ma, X. Miao, Z. Liu, L. Thiele, and Z. Yang, "Stitching weight-shared deep neural networks for efficient multitask inference on gpu," in *2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2022, pp. 145–153.

[54] J. Yi and Y. Lee, "Heimdall: mobile gpu coordination platform for augmented reality applications," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[55] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, "Band: coordinated multi-dnn inference on heterogeneous mobile processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022, pp. 235–247.

[56] Y. H. Oh, S. Kim, Y. Jin, S. Son, J. Bae, J. Lee, Y. Park, D. U. Kim, T. J. Ham, and J. W. Lee, "Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 584–597.

[57] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 320–333.

[58] J. Sevilla, L. Heim, M. Hobbhahn, T. Besiroglu, A. Ho, and P. Villalobos, "Estimating training compute of deep learning models," 2022, accessed: 2024-08-17. [Online]. Available: https://epochai.org/blog/estimating-training-compute

[59] Y. Ding, C. Niu, F. Wu, S. Tang, C. Lyu, and G. Chen, "Dc-ccl: Device-cloud collaborative controlled learning for large vision models," *arXiv preprint arXiv:2303.10361*, 2023.

[60] F. Yu, S. Bray, D. Wang, L. Shangguan, X. Tang, C. Liu, and X. Chen, "Automated runtime-aware scheduling for multi-tenant dnn inference on gpu," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[61] V. Lomonaco and D. Maltoni, "Core50: a new dataset and benchmark for continuous object recognition," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 17–26.

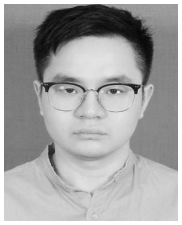
[62] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[63] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[64] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

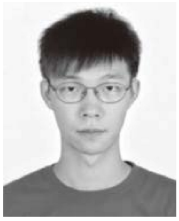
[65] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in neural information processing systems*, vol. 27, 2014.

[66] E. B. Zaken, Y. Goldberg, and S. Ravfogel, "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2022, pp. 1–9.
[67] J. Yao, F. Wang, K. Jia, B. Han, J. Zhou, and H. Yang, "Device-cloud collaborative learning for recommendation," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 3865–3874.
[68] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2938–2945.
[69] L. Du, A. T. Ho, and R. Cong, "Perceptual hashing for image authentication: A survey," *Signal Processing: Image Communication*, vol. 81, p. 115713, 2020.
[70] A. Forrest, "Colour histogram equalisation of multichannel images," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 152, no. 6, pp. 677–686, 2005.
[71] R. Csáky, P. Purgai, and G. Recski, "Improving neural conversational models with entropy-based data filtering," *arXiv preprint arXiv:1905.05471*, 2019.
[72] Y. Xin and S. Li, "Novel data-driven short-frequency mutual information entropy threshold filtering and its application to bearing fault diagnosis," *Measurement Science and Technology*, vol. 30, no. 11, p. 115006, 2019.
[73] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *2011 international conference on computer vision*. IEEE, 2011, pp. 991–998.
[74] M. Everingham and J. Winn, "The pascal visual object classes challenge 2012 (voc2012) development kit," *Pattern Anal. Stat. Model. Comput. Learn., Tech. Rep*, vol. 2007, no. 1-45, p. 5, 2012.
[75] NVIDIA, "Nvidia jetson orin nx," Product Introduction, Oct. 2023. [Online]. Available: https://www.nvidia.cn/autonomous-machines/embedded-systems/jetson-orin/
[76] AGILEX, "Smart car," Product Introduction, Oct. 2023. [Online]. Available: https://www.agilex.ai/
[77] LST, "Ladder-side-tuning," open code, Oct. 2023. [Online]. Available: https://github.com/ylsung/Ladder-Side-Tuning
[78] Pytorch, "Pytorch doc," Official Documentation, Oct. 2023. [Online]. Available: https://pytorch.org/docs/stable/nn.init.html#torch-nn-init
[79] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
[80] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi *et al.*, "Textbooks are all you need," *arXiv preprint arXiv:2306.11644*, 2023.
[81] C. Sakaridis, D. Dai, and L. Van Gool, "Acdc: The adverse conditions dataset with correspondences for semantic driving scene understanding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 765–10 775.
[82] H. Kang, Q. Hu, and Q. Zhang, "Sf-adapter: Computational-efficient source-free domain adaptation for human activity recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 7, no. 4, pp. 1–23, 2024.



Zhiqiang Cao received the master's degree in Electromagnetic Field and Microwave Technology from China radio propagation Research Institute, Qingdao, China, in 2020. He is currently pursuing the Ph.D. degree in computer science and technology with the Harbin Institute of Technology, Harbin, China. His research interests include computer vision, edge-cloud collaborative intelligence.

Yun Cheng received the B.Sc. and M.Sc. degrees in computer science and technology from Harbin Institute of Technology, Harbin, China, in 2012 and 2015, respectively.He received his Ph.D. degree from ETH Zürich in 2022. His research interests include machine intelligence and efficient, applied machine learning.



Zimu Zhou is currently a tenure-track assistant professor at the School of Data Science, City University of Hong Kong. He received his Ph.D. degree from Hong Kong University of Science and Technology in 2015. His research interests include model compression, federated machine learning, applied machine learning.

Youbing Hu received the master's degree in computer technology from Xidian University, Xi'an, China, in 2020. He is currently pursuing the Ph.D. degree in computer science and technology with the Harbin Institute of Technology, Harbin, China. His research interests include computer vision, edge-cloud collaborative intelligence, and continuous learning.

Anqi Lu received the master's degree in computer science and technology from Heilongjiang University, Harbin, China, in 2021. She is currently pursuing the Ph.D. degree in computer science and technology with the Harbin Institute of Technology, Harbin, China. Her research interests include computer vision and satellite-terrestrial collaborative intelligence.

Jie Liu (Fellow, IEEE) is a Chair Professor at Harbin Institute of Technology Shenzhen (HIT Shenzhen), China and the Dean of its AI Research Institute. Before joining HIT, he spent 18 years at Xerox PARC and Microsoft. He was a Principal Research Manager at Microsoft Research, Redmond and a partner of the company. His research interests are Cyber-Physical Systems, AI for IoT, and energy-efficient computing.

Ming Zhang received the bachelor's and Ph.D. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 1991 and 1997, respectively. He is currently a Distinguished Professor with the School of Computer Science and Technology, Soochow University, Suzhou, China. His current research interests include machine translation, natural language processing, and artificial intelligence.

Zhijun Li received the M.S. and Ph.D. degrees in computer science and technology from the Harbin Institute of Technology in 2001 and 2006, respectively. He is currently a Professor with the School of Computer Science and Technology, Harbin Institute of Technology. His research focuses on wireless networks, mobile computing, and artificial Internet of Things. He was a recipient of the MobiCom 2017 Best Paper Award.