EdgeMove: Pipelining Device-Edge Model Training for Mobile Intelligence

Zeqian Dong[2], Qiang He[1,2], Feifei Chen[3,*], Jin Hai[1], Tao Gu[4], Yun Yang[2]
[1]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]Department of Computing Technologies, Swinburne University of Technology, Australia
[3]School of Information Technology, Deakin University, Australia
[4]School of Computing, Macquarie University, Australia
zdong,qhe,yyang@swin.edu.au,feifei.chen@deakin.edu.au,tao.gu@mq.edu.au,hjin@hust.edu.cn

ABSTRACT

Training machine learning (ML) models on mobile and Web-of-Things (WoT) has been widely acknowledged and employed as a promising solution to privacy-preserving ML. However, these end-devices often suffer from constrained resources and fail to accommodate increasingly large ML models that crave great computation power. Offloading ML models partially to the cloud for training strikes a trade-off between privacy preservation and resource requirements. However, device-cloud training creates communication overheads that delay model training tremendously. This paper presents EdgeMove, the first device-edge training scheme that enables fast pipelined model training across edge devices and edge servers. It employs probing-based mechanisms to tackle the new challenges raised by device-edge training. Before training begins, it probes nearby edge servers' training performance and bootstraps model training by constructing a training pipeline with an approximate model partitioning. During the training process, EdgeMove accommodates user mobility and system dynamics by probing nearby edge servers' training performance adaptively and adapting the training pipeline proactively. Extensive experiments are conducted with two popular DNN models trained on four datasets for three ML tasks. The results demonstrate that EdgeMove achieves a 1.3×-2.1× speedup over the state-of-the-art scheme.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Parallel computing methodologies**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

KEYWORDS

Web of Things, edge computing, machine learning, edge intelligence

Qiang He and Hai Jin are also with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan, 430074, China.

WWW '23, May 1–5, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583540

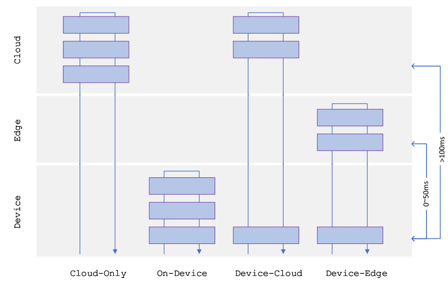


Figure 1: ML model training schemes.

ACM Reference Format:
Zeqian Dong[2], Qiang He[1,2], Feifei Chen[3,*], Jin Hai[1], Tao Gu[4], Yun Yang[2]. 2023. EdgeMove: Pipelining Device-Edge Model Training for Mobile Intelligence. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3543507.3583540

1 INTRODUCTION

Machine learning (ML) is powering an increasing variety of mobile and Web-of-Things (WoT) applications [20, 33], e.g., personalized recommendations [93], visual assistance [44], video analytics[32], etc. It is shifting ML model training and inference from the central cloud (i.e., cloud-only in Fig. 1) to the network edge [2, 45, 45, 46, 83, 92], enabling real-time data analytics and decision making with privacy preservation [4, 61, 74, 90].

A series of ML techniques have been proposed to support model training on mobile devices (i.e., On-Device in Fig. 1). For example, transfer learning can transfer the knowledge learned by an edge device's ML model to other edge devices' ML models to save on training expenses [6, 85]. Federated learning allows edge devices to train a model collaboratively without revealing their private training data [4]. However, the sizes of popular ML models have been increasing rapidly to pursue a higher model accuracy [55]. Many resource-constrained edge devices like smartphones and smart cameras cannot afford to train large models because they can be easily overwhelmed by the high computational overheads incurred [3, 9, 60, 88]. Many techniques, e.g., model compression [10], model pruning [47, 54], and sparse training [88], as well as tools, e.g., TensorFlow Lite [17] and PyTorch Mobile [1], have been developed to support model training on resource-constrained edge devices. However, it is difficult and often impossible to obtain a high model accuracy with these techniques [65].

An alternative solution is the device-cloud training scheme (see Fig. 1), which splits a model into two parts, one (and usually a small one) to be trained locally on the edge device and the other to be

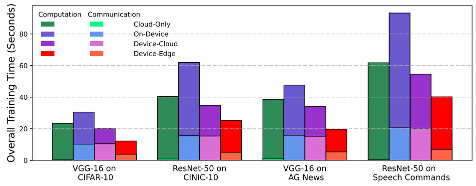


Figure 2: Training time under different schemes with a dedicated worker. Under Device-Cloud and Device-Edge, the client trains the lowest model layer and the worker trains the other model layers.

trained externally in the cloud [15, 61, 74, 87]. During the training process, every forward pass and backward pass goes through the edge device and the cloud back and forth. This training scheme allows edge devices to seek help from the cloud with training large ML models without revealing their private data. However, it is challenged by the unstable network conditions between edge devices and the cloud, which may vary during the training process and significantly impact training performance. Yao et al. developed a Device-Cloud scheme that re-partitions the model across the edge device and the cloud for training when the network condition varies [87]. However, under this scheme, model training is inevitably, cumulatively, and tremendously delayed by the communication latency between the edge device and the cloud. Fig. 2 compares the overall time taken to train four different ML models under the on-device scheme and the Device-Cloud scheme. We can easily see the extraordinary delay introduced by device-cloud communication under the Device-Cloud scheme, contributing to an average of 42.29% of the overall training time. It is also evidenced by the experimental results presented in [87]: a forward pass and a backward pass of a minibatch between an edge device and a cloud server take 100-500 milliseconds to complete. This price paid by the device-cloud training scheme to train large models is too high to be practical in most, if not all, real-world scenarios.

In an edge computing system powered by 5G, edge servers deployed at base stations or access points can take over a lot of workloads from various edge devices [2, 11, 22, 69, 76]. Within close geographic proximity, edge servers offer edge devices access to greater computation resources with single-digit millisecond communication latency [51]. ML models can be trained on edge servers to avoid excessive traffic over the backhaul network and enable fast model updates required by continual learning [2, 70]. Offloading the entire ML model from an edge device to a nearby edge server for training risks user privacy because private data will have to be uploaded to the edge server. Thus, this edge-only training scheme raises the same privacy concern as the cloud-only training scheme [4], which has raised wide privacy concerns in the real world [28]. In fact, many European cities specifically mandate against streaming users' data like video recordings to the cloud [2].

To systematically overcome the limitations of existing training schemes, this paper presents EdgeMove, a novel scheme that facilitates pipelined *device-edge* training (i.e., Device-Edge in Fig. 1). It splits a model into two partitions, one for training on the edge device and the other on an edge server. Intermediate features are transmitted in between instead of the user's private data. EdgeMove offers the following advantages over existing training schemes.

- Compared with the cloud-only training scheme and the edge-only training scheme [2], EdgeMove protects users' data privacy because it does not upload users' private data to the cloud or edge servers, which is subject to potential abuse, legal subpoenas, and extra judicial surveillance [71]. EdgeMove also protects users' model privacy because it offloads only part of a model for training, while the cloud-only training scheme and edge-only training scheme open the door to many cyber attacks [7, 58, 59, 73].
- Compared with the device-cloud training scheme [15, 61, 74, 87], EdgeMove significantly reduces the delays in the training process introduced by the communication latency between the local model part and the external (offloaded) model part. The key is to take advantage of the low-latency access to edge servers' computation resources. In Fig. 2, we can observe that training a model under EdgeMove (Device-Edge) takes much less time than under the device-cloud training scheme. Please note the this experiment is conducted under relatively stable network conditions without user mobility. In a real-world scenario where users move and network conditions vary, EdgeMove can achieve an even greater advantage in model training time (§5).

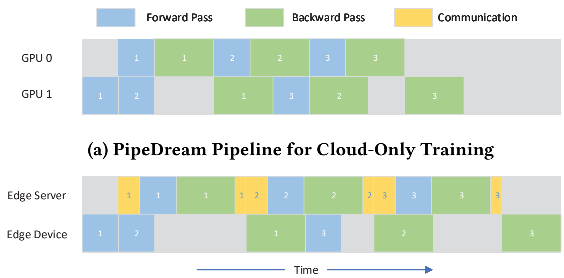
Contributions. Our work makes the following main contributions.

- EdgeMove is the first scheme devised to facilitate pipelined device-edge ML model training across edge devices and edge servers.
- In an edge computing system, an edge server's training performance replies on its GPUs available, its workloads, and the network condition, which often vary unpredictably over time. Model training strategies formulated offline can quickly be invalidated by these system dynamics. To tackle this challenge, EdgeMove probes the training performance of an edge device's nearby edge servers, and bootstraps model training with an approximate model partitioning (§3).
- During the training process, system resources often vary over time, sometimes mildly, sometimes considerably. For example, a more powerful GPU may become available. Edge servers' workloads may fluctuate. Network conditions may also change. These variations are usually more drastic when the user moves. EdgeMove probes nearby edge servers' training performance adaptively and adapts the training pipeline proactively according to system dynamics at runtime (§4).
- To evaluate the performance of EdgeMove, extensive experiments are conducted in an edge computing system with 2 popular ML models trained on 4 datasets for 3 types of ML tasks. The results demonstrate EdgeMove's superior performance over existing training schemes. Compared with the state-of-the-art training scheme, EdgeMove achieves a speedup of 1.5×-2.1× in training ML models for image classification, 1.2×-2.0× for text classification, and 1.2×-1.8× for audio classification.

2 BACKGROUND AND CHALLENGES

2.1 Parallelisms for ML Model Training

A series of parallelism schemes have been proposed to accelerate ML model training. Data parallelism partitions training data across multiple workers [5, 40, 41, 80]. Specifically, training data is partitioned into multiple subsets, one for each worker, so that workers can train their ML model replicas in parallel. To accommodate the increasingly large and complex ML models, model parallelism partitions an ML model across multiple workers [13, 27]. It allows each worker to their model partitions individually and the entire model collectively. Data parallelism and model parallelism can also



(a) PipeDream Pipeline for Cloud-Only Training

(b) PipeDream Pipeline for Device-Edge Training

Figure 3: PipeDream pipeline across two GPUs in a cluster (cloud-only training) vs. across an edge device and an edge server (device-edge training). In this example, a backward pass takes twice as long as a forward pass because it is more computationally expensive. Uplink transmission takes twice as long as downlink transmission.

be integrated to combine their advantages [27, 29–31, 43, 55]. In recent years, pipeline parallelism has demonstrated significant advantages over data parallelism and model parallelism by leveraging intra-batch parallelism and inter-batch parallelism. Combining model parallelism and intra-batch parallelism, GPipe splits a mini-batch into multiple micro-batches and performs forward passes followed by backward passes for these micro-batches across multiple pipelined workers [27]. Taking a step further, PipeDream integrates inter-batch parallelism into the pipeline, allowing the forward and backward passes of minibatches to partially overlap [55]. Compared with GPipe, it accelerates model training significantly by reducing pipeline flushes. However, PipeDream was designed to pipeline ML model training across workers in the same clusters where workers are connected with high-speed interconnects like NVLink (25 Gpbs) and PCIe (10 Gbps). Its pipeline does not consider the communication latency between workers, as shown in Fig. 3a. When it is applied in device-edge training, the communication latency between the edge device and the edge server undermines the pipeline parallelism and slows down the training process significantly. To overcome this limitation, EdgeMove considers device-edge communication latency when partitioning an ML model for pipeline deployment across the edge device and the edge server (§3.2).

2.2 Edge Computing System Dynamics

Most existing pipeline schemes, e.g., PipeDream [55], GPipe [27], HetPipe [64], and PTD-P [57], are designed for ML model training in GPU clusters. Compared with GPU clusters, edge computing systems exhibit unique characteristics that challenge the application of existing pipeline schemes to device-edge ML model training.

Network Conditions. GPUs/workers[1] in a cluster are linked with high-speed interconnects while communications between edge devices and edge servers are much slower (§2.1).

GPU Resources. Most pipeline schemes assume that the GPUs available for ML model training are virtually unlimited. For example, PipeDream and GPipe partition ML models for training on as many GPUs as needed to maximize pipeline throughput [27, 55]. PTP-P goes to extremes and partitions an ML model for training across thousands of GPUs [57]. In an edge computing system, an edge device can only access a nearby edge server covering the edge device [24, 66, 84]. The computation resources, e.g., CPUs, memory, and GPUs, available on its nearby edge servers are limited due to their small physical sizes[2, 38, 89]. In addition, it is unknown to the edge device what GPUs and how of many of them are available on nearby edge servers for training its ML model. Furthermore, an edge server's training performance relies on not only the specs of the GPUs available but also its workloads and the network condition. For example, a fully loaded edge server may not be able to guarantee its training performance as expected. While many pipeline schemes see GPU clusters as white boxes, the edge devices in an edge computing system have to consider edge servers as black boxes, particularly when serverless computing is implemented to relieve edge devices of resource allocation and server management [14, 42, 63, 67, 75]. EdgeMove tackles this challenge by probing the end-to-end training performance of an edge device's nearby edge servers at the beginning of the training process for pipeline construction (§3).

Runtime Dynamics. Most existing pipeline schemes do not adapt model partitionings during the entire training process, assuming that GPUs' training performance always remains stable. This may be true in a dedicated GPU cluster but is unrealistic in edge computing systems. An edge server's resources are shared by multiple tenants like YouTube and Uber [24, 37, 68]. The availability of its GPUs may vary during the training process. Its workloads and network conditions may also vary during the training process, depending on the user demands within its coverage. These system dynamics can undermine or improve edge servers' runtime training performance, especially when the edge device moves. To tackle this challenge, EdgeMove monitors the end-to-end training performance of the worker, i.e., the edge server training the ML model at the moment, probes the end-to-end performance of nearby edge servers adaptively, and adapts the training pipeline proactively (§4).

2.3 Device-Cloud ML Model Training

Yao et al. proposed the first scheme to enable device-cloud training across an edge device and a cloud server [87]. Their scheme consists of an offline phase and an online phase. In the offline phase, the edge device evaluates its own training performance, the cloud server's training performance, and the network conditions. Then, Device-Cloud splits the ML model into two partitions, one to be trained locally and the other to be trained by the cloud server. The experimental results presented in [87] reveal that the network condition is critical to the end-to-end training performance. Thus, the key idea of Device-Cloud is to find the partition point that minimizes the time taken to transmit intermediate features between the edge device and the cloud server. Similar to EdgeMove, Device-Cloud recognizes the potential changes in the network conditions during the training process. In the online phase, it monitors the uplink and downlink data rate, and evaluates the training pipeline. If it is suboptimal, Device-cloud enumerates all the partition points based on the current network conditions and adapts the training pipeline accordingly. Device-cloud is not suitable for device-edge training for two main reasons.

- Device-Cloud assumes a single powerful cloud server as the external worker with fixed specs. The experimental results presented in [87] suggest offloading as many model layers as possible to the cloud for training. However, in an edge computing system, edge servers are heterogeneous and their available resources vary over

[1]This paper speaks of "GPU" and "worker" interchangeably if the worker is a GPU.

time [2, 21, 25]. It is not always the optimal solution to offload the most model layers for training. In addition, as shown in Fig. 3, device-edge communication latency factors into the performance of the device-edge pipeline. To optimize the pipeline performance, EdgeMove sees edge servers as black boxes, and evaluates their end-to-end training performance, taking into account the computation dimension and the communication dimension in both training pipeline construction (§3) and adaptation (§4).

- Similar to its offline training pipeline construction, Device-Cloud's online training pipeline adaptation only considers the communication dimension, i.e., the changes in uplink and downlink data rates. However, the dynamics in edge servers' computation resources must also be considered at runtime, especially when the clients move (§2.1). In addition, Device-Cloud re-evaluates network conditions and pipeline performance at every training iteration. This incurs excessive (and often unaffordable) resource consumption on edge devices, and more so in an edge computing system where multiple edge servers may be available as the external worker. Taking both computation and communication into account, EdgeMove monitors the current edge server's end-to-end training performance at runtime by training cycles (1 cycle = 500 training iterations) (§3), probes nearby edge servers' end-to-end training performance when it is necessary, and adapts the training pipeline when it is worthwhile (§4).

To enable device-edge training, EdgeMove consists of two phases, i.e., training pipeline construction (§3) and adaptation (§4). It may complement existing model training schemes. For example, PipeDream and GPipe can be implemented on edge servers to accelerate the training of model partitions offloaded from clients under EdgeMove.

3 OFFLINE PIPELINE CONSTRUCTION

Similar to pipelined cloud-only training [27, 55, 57, 64], pipelined device-cloud training and device-edge training both require splitting the target ML model into two partitions, one for local training and the other for external training. The key is to find the optimal partitioning that maximizes the pipeline throughput (and minimizes the training time). Existing pipeline schemes profile the target model before model partitioning, assuming that external workers' training performance is known and unchanged (§2.2). For example, PipeDream [55] obtains the estimates of each model's compute time and output size based on the performance of the GPUs. Then, it employs dynamic programming to find the optimal partitioning progressively from the lowest model layer to the highest. Device-Cloud [87] also profiles the target model layer by layer but on the edge device and the cloud server. Then, it runs tests, layer by layer, through the device-cloud pipeline to find the optimal partitioning. These sophisticated profiling and partitioning techniques are unsuitable for device-edge training because the system dynamics can quickly invalidate the previously-optimal partitioning and undermine the pipeline performance (§2.1). An illustrative example can be found in Appendix B. In addition, Device-Cloud uploads the entire target model to the cloud server for profiling, which reveals the model architecture and opens the door to cyber attacks (§1).

3.1 Model Profiling

EdgeMove also profiles the target model M on the client and its nearby edge servers. It sends a test model M_t and a test dataset D_t to each nearby edge server for training. When they complete the training cycle (§4) and return the results, the client obtains the estimates of their end-to-end training performance, measured by the time taken to complete a training epoch plus the RTT (Round Trip Time). The edge server with the best performance is selected as the *worker*. In the meantime, other nearby edge servers register the client's training task. When they can improve their training performance with a more powerful GPU or more pipelined GPUs, they notify the client, which may activate a training pipeline adaptation (§4). This way, the client does not constantly have to probe nearby edge servers' available GPUs at runtime.

Unlike PipeDream or Device-Cloud, EdgeMove does not transmit the entire model to external workers to probe their training performance. Instead, it produces the testing models by obfuscating the model parameters[2]. This protects the edge device from many cyber attacks [7, 58, 59, 73] by preventing the reveal of the true model architecture. The test dataset D_t comprises randomly generated samples to protect the user's data privacy. For example, a dummy image generator[3] is used to generate dummy images for profiling image classification models in our study.

3.2 Model Partitioning

As discussed in §3, system dynamics may quickly invalidate the optimal model partitioning found offline. Thus, knowing the impracticality of the sophisticated model partitioning techniques employed by PipeDream and Device-Cloud in edge computing systems, EdgeMove employs a lightweight partitioning technique to bootstrap pipelined device-edge model training. **Its main idea is to commence model training rapidly with an approximate partitioning, followed by online adaptation (§4.2).** Let t_d denote the time taken by the edge device to complete a training cycle in model profiling, S denote the set of nearby edge servers, and $t_{e,n}$ denote the time taken by the n-th nearby edge server in S. We calculate their per-layer training time as follows:

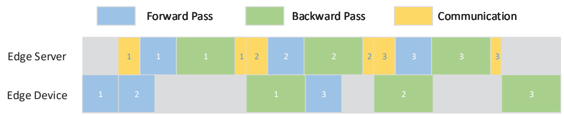
$$\begin{cases} \overline{t_d} = \frac{t_d}{R \times L} \\ \overline{t_{e,n}} = \frac{t_{e,n}}{R \times L}, \forall s_n \in S \end{cases} \tag{1}$$

where R is the number of training iterations in a training circle (§4), and L is the number of layers in the target model M.

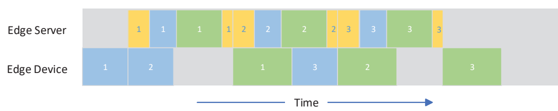
The objective of model partitioning under EdgeMove is to maximize the pipeline throughput, similar to existing pipeline schemes. The key is *throughput balance*, i.e., for the edge device and the edge server to sustain roughly the same throughput. Noting that the edge device's training performance is lower than the edge server, EdgeMove partitioning objective is to ensure that the time taken to process a minibatch (including forward and backward passes) roughly equals to the time taken to complete the backward pass of the previous minibatch plus the forward pass of the next minibatch. Take Fig. 4 for example, which compares an example EdgeMove pipeline with a PipeDream pipeline for device-edge training. There should be $t_{ul}(b_2) + t_{fp}(b_2) + t_{bp}(b_2) + t_{dl}(b_2) = t_{bp}(b_1) + t_{fp}(b_3)$, where ul stands for uplink, dl stands for downlink, fp stands for forward pass, and bp stands for backward pass.

[2] Many off-the-shelf techniques can be employed. In our experiments, we employ Neurobfuscator [39] available at https://github.com/zlijingtao/Neurobfuscator.

[3] https://github.com/FabianBeiner/PHP-Dummy-Image-Generator



(a) PipeDream Pipeline



(b) EdgeMove PipeLine

Figure 4: PipeDream vs. EdgeMove for device-edge training.

EdgeMove employs Algorithm 1 (in Appendix) to find an approximate partitioning for model M. Based on the result, M can be split into two model partitions, M_d for the client and M_e for the worker. They are deployed accordingly to establish a device-edge training pipeline. This concludes offline pipeline construction and training commences.

4 ONLINE PIPELINE ADAPTATION

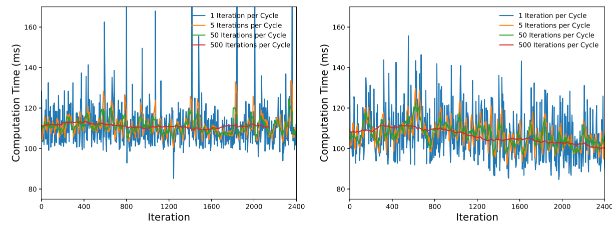
At runtime, EdgeMove monitors the pipeline performance and adapts the training pipeline accordingly. It may adapt the training pipeline in two ways: 1) adapting model partitioning across the client and the worker; 2) adapting the entire training pipeline with a new worker and potentially a new model partitioning.

4.1 Pipeline Monitoring

At runtime, system dynamics like changes in the network conditions and worker's workload conditions may undermine the performance of the pipeline §2.2. To optimize the pipeline, the client monitors the worker's training performance by inspecting its end-to-end training time, and adapts the training pipeline accordingly. For example, if the worker is taking more time than before to train its model partition, it may disrupt the throughput balance between the client and the worker (§3) and undermine the pipeline performance. To restore the throughput balance, one or more model layers need to be transferred from the worker to the client.

The training epoch can conduct the inspection (i.e., after every training epoch), similar to how many techniques adapt model training [26, 34, 49, 82]. However, pipeline adaptation by the epoch (i.e., the processing of all the minibatches) is unsuitable for device-cloud training. It is an overly low granularity. Take VGG-16 [72] for example. It takes an RTX3080Ti GPU 24 seconds to complete a training epoch on the CIFAR-10 dataset [35] with a batch size of 32. In device-edge training, during such a long period, the system dynamics may have already undermined the pipeline performance immensely.

Device-Cloud [87] goes to another extreme by inspecting network conditions by the training iteration (i.e., after processing each minibatch). This granularity is too high for device-edge training. Fig. 5 shows the per-iteration computation time (1 Iteration per Circle) for training a VGG-16 model and a ResNet-50 model. We can see significant and constant fluctuations across different training iterations. There are various possible causes, e.g., the worker's workload dynamics, the model's inner nature, and sample characteristics. These fluctuations easily activate excessive pipeline adaptations, which require pipeline flushes, model re-partitioning, and model



(a) VGG-16 on CIFAR-10 **(b) ResNet-50 on Speech Command**

Figure 5: Per-circle rolling average computation time for model training. When the circle size is 5 iterations, the rolling average takes the last 5 iterations, adds them up, and divides the sum by 5. a) VGG-16 is trained on a RTX3080Ti GPU with a batch size of 16. b) ResNet-50 is training on the same GPU with a batch size of 16.

partition re-deployment. Frequent pipeline adaptations will cause system oscillation and decelerate model training. In the meantime, it may incur enormous computation overheads for the client.

EdgeMove inspects pipeline performance at runtime by the *training cycle*. A training cycle consists of an appropriate number of training iterations, not too many or too few. To find the right number of training iterations, we set the cycle size at 5, 50, and 500 and include the corresponding per-cycle rolling average computation time in Fig. 5. When the circle size is 5 iterations, the rolling average still fluctuates significantly and continuously (with a coefficient of variation of 0.137). When it increases to 500 iterations, the rolling average training time stabilizes for both models with a coefficient of variation of only 0.029. Inspecting pipeline performance after every 500 iterations, EdgeMove can minimize the uncertainties caused by computation time fluctuations in an edge server's end-to-end training performance and focus on the communication dimension, i.e., network condition changes. In addition, an appropriate circle size allows the client to inspect pipeline performance and adapt the pipeline on time without excessive computation overheads. A different ML model, a different training dataset, or a different batch size may require a different circle size. It can be easily obtained by profiling the target ML model offline. In our experiments, three different circle sizes are used (§5).

4.2 Partitioning Adaptation

As discussed in §3.2, EdgeMove employs a lightweight technique to find an approximate model partitioning for bootstrapping model training. After training commences, EdgeMove starts monitoring pipeline performance for pipeline adaptation (§4.1). One of EdgeMove's key online tasks is adapting the approximate model partitioning to optimize pipeline performance. It employs a novel partition adaptation technique to achieve this objective. The main idea is to move the partition point by one layer (to a lower layer or a higher layer) after each training circle based on pipeline performance changes until the optimal partition point is found. It continues cycle-by-circle over the entire session with the worker until worker adaptation (§4.3). The pseudocode is presented in Appendix 2

EdgeMove runs the algorithm when training commences. Let c_1 denote the first training circle. At the end of c_1, the algorithm measures the end-to-end training time (i.e., the overall time for the client and the worker to complete c_1), and moves partition point randomly by one layer, across either the previous layer or the next layer (Line 2-3). Please note that if the partition point is after the first layer, it moves across the next layer. At the end of c_2, the algorithm again measures the end-to-end training time and compares it with the performance in c_1 (Line 10-12). If it takes less time in c_2 than in c_1, the movement of the partition point at the end of c_1 was correct. The algorithm will move the partition point across one layer in the same direction to approach to optimal partition point (Line 7). Otherwise, it moves the partition point across one layer in the opposite direction (Line 11). This continues until the end-to-end training time stabilizes, which indicates that EdgeMove found the optimal partition point for balancing the client and the worker's throughput (§3.2). Please note that the partition point may move back and forth around the optimal partition point. To avoid such movements, the algorithm will move the partition point only when the expected performance difference is more significant than those caused by previous movements in the same direction.

4.3 Worker Adaptation

By balancing the client and the worker's throughput, partitioning adaptation can cope with mild variations in pipeline performance. However, when the pipeline performance decreases drastically, partitioning adaptation may struggle to ensure pipeline performance. The client and the worker can both contribute to a drastic pipeline performance decrease. For example, when the client switches to an energy-saving mode, e.g., Android's "Battery Saver" mode [18], its computation power declines. The decline could be so significant that even offloading most model layers to the worker through partitioning adaptation (§4.2) cannot restore pipeline performance to a satisfactory level. Similarly, when the worker's end-to-end performance degrades drastically due to unexpected runtime events like network condition deterioration, workload bursts, cyber attacks and client movement, training the fewest model layers on the worker may not be able to ensure the pipeline performance either. It also violates the objective of device-edge training, i.e., to leverage edge servers' computation power. EdgeMove tackles these challenges with an adaptive worker adaptation technique that finds a new worker to replace the previous one. The pseudocode is presented in Appendix 3

Worker adaptation requires probing nearby edge servers' end-to-end training performance online with model profiling (§3.1). The client can monitor pipeline performance constantly (§4.1) because it consumes little computation resources. However, it is impractical for the client to probe nearby edge servers constantly. Under EdgeMove, the client records the worker's worst end-to-end training performance at the end of every training circle (§4.2), denoted by t_{wst}. At the end of each training cycle, the client inspects the worker's performance against t_{wst} to determine whether it is necessary to probe nearby edge servers' training performance. When the worker's training performance is worse than t_{wst}, the client probes nearby edge servers' training performance (including the worker's) in the same way as offline model profiling (§3.2). In the meantime, model training proceeds to the next circle, asynchronously, with the online probing process.

When the probing results come back from the nearby edge servers, the client compares their training performance with the worker's performance obtained at the end of the current training circle. If the worker is outperformed, the client selects the edge server with the best training performance as the new worker. Then, it partitions the ML model and builds a new training pipeline with the technique presented in §3.2. If the worker is not outperformed by any other nearby edge server, the client may retain the current training pipeline, wait for the next opportunity, or simply terminate the training process.

When profiling a model, the client registers its training task with probed edge servers (§3.1). If an edge server's training performance improves at runtime, it can notify the client. This may also activate a worker adaptation, which compares the edge server's performance against the worker's.

Remark. Worker adaptation incurs migration overheads. The client needs to terminate the training pipeline and build a new one. These take time. However, in the long term, these once-off overheads are worthwhile compared with the performance gains from training with a more powerful new worker. In addition, a pub-sub system is needed to allow the client to register its training task with probed edge servers. Such a system can be built easily based on pub/sub services offered by Google [19], Microsoft [53], Amazon [52], etc. It barely consumes edge servers' resources and can benefit other applications running in the edge computing system. Without this pub-sub system, the client has to wait until it probes a nearby edge server to discover its improved training performance. This slightly reduces EdgeMove's responsiveness to system dynamics but does not fundamentally undermine its usefulness.

5 EVALUATION

5.1 Experiment Setup

System Setup. An edge computing system is built with 125 virtual machines deployed in a private data center as edge servers with a coverage radius of [300, 500] meters, each with a 4-core vCPU, 8-16GB RAM, and a 2080Ti GPU or a 3080Ti GPU. These virtual machines are each assigned a location extracted from the widely-used EUA dataset[4] [36] to simulate the Melbourne CBD area powered by edge computing. An Amazon p3.2xlarge EC2 instance with 8 vCPUs, 61GB RAM and a V100 GPU is hired as the cloud server to enable Device-Cloud [87]. A Google Pixel 6a connects to the system via a Wi-Fi 6 router as the client. We ran tests on the system and measured the network conditions. The network latency between the client and the cloud server is [100, 170] milliseconds. The network latency between the client and the edge servers is [5-40] milliseconds, similar to the 5G network latency in the U.S. reported by Ericsson in August 2022 [48]. At runtime, an edge server may be able to improve its training performance at runtime with a more powerful GPU or more pipelined GPUs (§3.1). From the client's perspective, it is no different from discovering a new edge server nearby. Thus, in the experiments, edge servers' GPU resource dynamics are not implemented.

Client Movement. In the experiments, the client moves along one of the ten randomly-created trajectories across the Melbourne CBD. These trajectories include different numbers of turns at different locations. Two example trajectories are illustrated in Appendix C. When the client moves, its nearby edge servers are identified based on the distance between them - those that cover the client are considered its nearby edge servers.

[4]https://github.com/swinedge/eua-dataset

Models and Datasets. VGG-16 [72] is trained on the CIFAR-10 dataset [16] for image recognition, and on the AG News dataset [91] for text classification. ResNet-50 [23] is trained on the Speech Commands dataset [81] for speech recognition, and on the CINIC-10 dataset [12] for image recognition. More details about the models and the datasets can be found in Appendix D.

Baseline. EdgeMove is evaluated against three representative training schemes, including a baseline and two state-of-the-art schemes.

- **On-Device [87].** Under this baseline scheme, ML models are trained only on the client's edge device.
- **Device-Cloud. [87]** This is the state-of-the-art scheme for device-cloud training. In the experiments, it builds training pipelines across clients and the cloud server. First, it profiles the target model layer by layer, similar to PipeDream, and finds the optimal model partitioning for device-cloud training, considering both computation and communication conditions. At runtime, it monitors the network conditions and adapts model partitioning based on uplink and downlink data rates without worker adaptation. A detailed discussion about this scheme can be found in §2.3.
- **PipeDream-E [55].** PipeDream is the state-of-the-art scheme for pipelining ML model training across GPUs in a cluster. Several schemes have adapted PipeDream to various ML training scenarios [56, 57, 86]. These schemes are not implemented in the experiments because they share the same core idea with PipeDream. PipeDream-E adapts PipeDream to device-edge training by building training pipelines across clients and edge servers. The experiments randomly select one of the client's nearby edge servers as the worker offline (and online when the client leaves the worker's coverage area). Then, it profiles the model layer by layer and finds the optimal model partitioning across the client and the worker without considering device-edge network conditions. After that, it builds a training pipeline across the client and the worker without partitioning adaptation or worker adaptation. More details about PipeDream can be found in §2.1.

Performance Metrics. Acceleration is the main objective of pipeline training [27, 55, 87]. In the experiments, we evaluate EdgeMove's training time-to-accuracy, i.e., the time taken to train a model to the target accuracy, same as [27, 55, 87].

5.2 Experimental Results

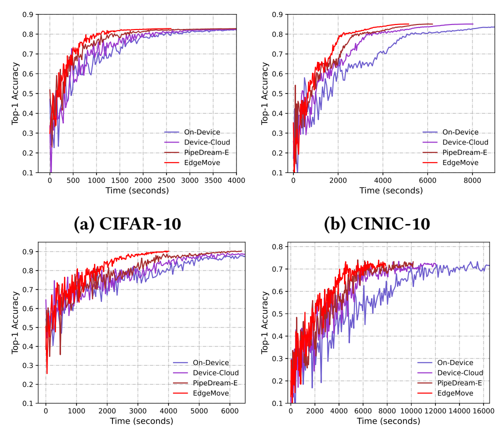
Overall Performance. Table 1 summarizes the average time taken for the clients to train a model to the target accuracy under different schemes, as well as their speedups over On-Device. The following key findings can be derived from the table.

- On-Device takes the most time to train a model in all the cases. This is not surprising because it does not leverage powerful workers or pipeline parallelism to accelerate model training.
- Powered by pipeline parallelism and external workers, PipeDream-E and Device-Cloud both reduce the training time considerably, evidenced by their significant speedups over On-Device. This confirms the effects of pipelining model training across the client and a powerful external worker.
- PipeDream-E and EdgeMove both outperform Device-Cloud evidently. Their advantages come from the much lower device-edge communication latency compared with device-cloud communication latency (§1). This indicates the fundamental advantage of device-edge training over device-cloud training.
- EdgeMove outperforms PipeDream-E remarkably. The main reason is that it factors into three elements that are neglected by PipeDream-E: 1) edge servers' heterogeneous GPU resources; 2) network conditions between the client and different edge servers; and 3) runtime system dynamics in both computation and communication. This tells us that EdgeMove can properly tackle the challenges discussed in §2.

Model Convergence. Fig. 6 illustrates clients' model convergence under different training schemes. Despite the models and the datasets, we can see that EdgeMove is always the first to complete the training process, with much time to spare compared with its competitors. Looking closely at the early stage of the training process, we can see that EdgeMove's model accuracy increase is relatively slow, even slower than PipeDream-E. About 50-75 seconds in, the increase rate starts to pick up. This is EdgeMove's "warm-up" phase, where partitioning adaptation (§3.2) kicks in to find the optimal partitioning. After the warm-up phase, EdgeMove's model accuracy increase remains well above every other training scheme, making it always the first to converge the model.

Table 1: Training time-to-accuracy (seconds) and speedups over On-Device. The highest performance in each column is underlined.

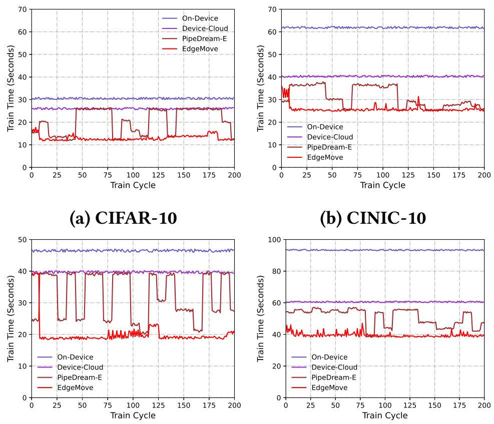
Model & Dataset	Scheme	Training Time	Speed Up
VGG-16 CIFAR-10 Target Accuracy: 82.8%	On-Device	3,053	1.00×
	Device-Cloud	2,601	1.17×
	PipeDream-E	1,937	1.58×
	EdgeMove	1,283	2.38×
ResNet-50 CINIC-10 Target Accuracy: 85.0%	On-Device	6,188	1.00×
	Device-Cloud	4,027	1.54×
	PipeDream-E	3,263	1.90×
	EdgeMove	2,626	2.36×
VGG-16 AG News Target Accuracy: 90.0%	On-Device	4,645	1.00×
	Device-Cloud	3,967	1.17×
	PipeDream-E	3,490	1.33×
	EdgeMove	2,002	2.32×
Resnet-50 Speech Commands Target Accuracy: 72.8%	On-Device	9,331	1.00×
	Device-Cloud	6,052	1.54×
	PipeDream-E	5,118	1.82×
	EdgeMove	3,988	2.33×



(a) CIFAR-10 (b) CINIC-10

(c) AG News (d) Speech Commands

Figure 6: Model convergence.



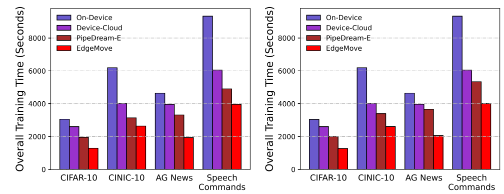
(a) CIFAR-10 (b) CINIC-10

(c) AG News (d) Speech Commands

Figure 7: Per-circle training time in the first 200 training circles.

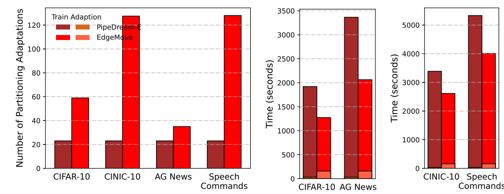
Per-circle Training Time. Compared with PipeDream-E and Device-Cloud, one of EdgeMove's unique features is its consideration of edge servers' end-to-end training performance, treating them as black boxes (§2). It is the basis for the model profiling (§3.1) and pipeline monitoring (§4.1) mechanisms designed for EdgeMove. To validate the usefulness of this feature, Fig. 7 demonstrates the clients' average per-circle training time in the first 200 training circles. As expected, we can observe that EdgeMove's per-circle training time is much lower than its competitors. In the beginning, it takes more time to complete a training circle than PipeDream-E. This confirms that the approximate model partitioning (§3.2) EdgeMove obtains offline to bootstrap model training is indeed not optimal. Luckily, through partitioning adaptation (§4.2), EdgeMove manages to find the optimal model partitioning across the client and the worker rapidly. This is evidenced by the quick decrease in its per-circle training time after model training commences. In Fig. 7, we can observe peaks in EdgeMove's per-circle training time, as well as other training schemes, some larger than others. We investigated and found that most of the small ones were caused by partitioning adaptation (§4.2), while most of the large ones were caused by worker adaptation (§4.3). We can also observe peaks in PipeDream-E's per-circle training time, which appear when the client is no longer within the worker's coverage and has to find a new worker. EdgeMove's peaks are much smaller than PipeDream-E's on average. This indicates the ability of EdgeMove to adapt partitioning or worker timely in response to system dynamics.

Adaptation to Client Speed. The clients in an edge computing system may move at different speeds. For example, they may travel in different ways, on foot, by bicycle or car. System dynamics are usually more significant when a client is travelling at a higher speed. To evaluate EdgeMove's ability to adapt to client speed, we conduct an experiment where clients move along the same trajectories at twice the speed in the previous experiments. Fig. 8 compares the training time-to-accuracy in these two cases. We can see that when the clients move faster, On-Device and Device-Cloud take almost the same time to train their models. They do not have to adapt to client speed because it does not impact on-device training or device-cloud training. PipeDream-E's performance is much worse, taking an average of 44.2% more time to train clients' models across the entire experiment. EdgeMove wins this contest easily against



(a) Slow Clients (b) Fast Clients

Figure 8: Training time-to-accuracy when different client speeds.



(a) (b) (c)

Figure 9: Adaptation overheads.

PipeDream-E. We can barely observe any increases in its training time when clients speed up. This shows that EdgeMove can properly accommodate user mobility, which is a grand challenge in a lot of studies on edge computing [8, 50, 62, 77–79].

Adaptation Overheads. Partitioning adaptation (§4.2) and worker adaptation (§4.3) incur overheads, measured by the time taken to upload new model layers to the worker and the time taken to upload model partitions to new workers. Fig. 9a shows the average number of partitioning adaptations EdgeMove needs to train a model and compares the number of worker adaptations it needs against PipeDream-E. PipeDream-E replaces the worker with a new worker *reactively* only when it leaves the worker's coverage area (§5.1). To respond to system dynamics timely, EdgeMove replaces the worker *proactively* based on its end-to-end training performance against other nearby edge servers' (§4). Thus, it evidently needs more worker adaptations than PipeDream-E. Fig. 9c compares EdgeMove's adaptation time (i.e., time for adaptation) and training time (overall training time minus adaptation time) against PipeDream-E's. We can see that EdgeMove spends more time adapting than PipeDream. However, its overall training time is much less than PipeDream-E. This indicates that its proactive adaptations pay off with significant speedup gains.

6 CONCLUSION AND FUTURE WORK

This paper presented EdgeMove, a novel scheme that pipelines device-edge machine learning (ML) model training. To tackle the challenges raised by the edge computing system dynamics, it bootstraps model training with an approximate training pipeline constructed based on edge servers' end-to-end training performance. At runtime, it monitors pipeline performance and adapts the training pipeline accordingly. Compared with state-of-the-art schemes, EdgeMove completes model training up to 2.4× faster across a range of ML tasks.

ACKNOWLEDGMENTS

This research is partially funded by Australian Research Council Discovery Projects (DP180100212 and DP200102491). Feifei Chen is the corresponding author of this paper.

REFERENCES

[1] Meta AI. Pytorch mobile, 2021.
[2] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation*, pages 119–135, 2022.
[3] Ilai Bistritz, Ariana Mann, and Nicholas Bambos. Distributed distillation for on-device learning. *Advances in Neural Information Processing Systems*, 33:22593–22604, 2020.
[4] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.
[5] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *19th International Conference on Computational Statistics*, pages 177–186, 2010.
[6] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, 33:11285–11297, 2020.
[7] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *IEEE security and privacy workshops*, pages 1–7. IEEE, 2018.
[8] Jun Lin Chen, Daniyal Liaqat, Moshe Gabel, and Eyal de Lara. Starlight: Fast container provisioning on the edge and over the {WAN}. In *19th USENIX Symposium on Networked Systems Design and Implementation*, pages 35–50, 2022.
[9] Min Chen, Haichuan Wang, Zeyu Meng, Hongli Xu, Yang Xu, Jianchun Liu, and He Huang. Joint data collection and resource allocation for distributed machine learning at the edge. *IEEE Transactions on Mobile Computing*, 2020.
[10] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294. PMLR, 2015.
[11] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM TON*, (5):2795–2808, 2016.
[12] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. CINIC-10 is not imagenet or CIFAR-10. *arXiv preprint arXiv:1810.03505*, 2018.
[13] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in Neural Information Processing Systems*, 25:1223–1231, 2012.
[14] Shuiguang Deng, Hailiang Zhao, Zhengzhe Xiang, Cheng Zhang, Rong Jiang, Ying Li, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Dependent function embedding for distributed serverless edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(10):2346–2357, 2021.
[15] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 20(2):565–576, 2019.
[16] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
[17] Google. Tensorflow lite, 2017.
[18] Google. Use battery saver on a pixel phone, 2022.
[19] Google. What is pub/sub?, 2022.
[20] Peizhen Guo, Bo Hu, and Wenjun Hu. Mistify: Automating dnn model porting for on-device inference at the edge. In *18th USENIX Symposium on Networked Systems Design and Implementation*, pages 705–719, 2021.
[21] Rui Han, Shilin Li, Xiangwei Wang, Chi Harold Liu, Gaofeng Xin, and Lydia Y Chen. Accelerating gossip-based deep learning in heterogeneous edge computing platforms. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1591–1602, 2020.
[22] Samira Hayat, Roland Jung, Hermann Hellwagner, Christian Bettstetter, Driton Emini, and Dominik Schnieders. Edge computing in 5g for drone navigation: what to offload? *IEEE Robotics and Automation Letters*, 6(2):2571–2578, 2021.
[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
[24] Qiang He, Guangming Cui, Xuyun Zhang, Feifei Chen, Shuiguang Deng, Hai Jin, Yanhui Li, and Yun Yang. A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 31(3):515–529, 2020.
[25] Hanpeng Hu, Dan Wang, and Chuan Wu. Distributed machine learning through heterogeneous edge systems. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 7179–7186, 2020.
[26] Lang Huang, Chao Zhang, and Hongyang Zhang. Self-adaptive training: beyond empirical risk minimization. *Advances in Neural Information Processing Systems*, 33:19365–19376, 2020.
[27] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems*, 32, 2019.
[28] David Jeans. Related's hudson yards: Smart city or surveillance city? *The Real Deal*, 15, 2019.
[29] Zhihao Jia, Sina Lin, Charles R Qi, and Alex Aiken. Exploring hidden dimensions in parallelizing convolutional neural networks. In *28 International Conference on Machine Learning*, pages 2279–2288, 2018.
[30] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.
[31] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. Heterogeneity-aware distributed parameter servers. In *ACM International Conference on Management of Data*, pages 463–478, 2017.
[32] Avi Kewalramani. Live video analytics with microsoft rocket for reducing edge compute costs, 2020.
[33] Young Geun Kim and Carole-Jean Wu. Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1082–1096. IEEE, 2020.
[34] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K Bansal, William Constable, Oguz Elibol, Scott Gray, Stewart Hall, Luke Hornof, et al. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.
[35] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
[36] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, and Yun Yang. Optimal edge user allocation in edge computing with variable sized vector bin packing. In *International Conference on Service-Oriented Computing*, pages 230–245. Springer, 2018.
[37] Bo Li, Qiang He, Feifei Chen, Hai Jin, Yang Xiang, and Yun Yang. Auditing cache data integrity in the edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1210–1223, 2021.
[38] Bo Li, Qiang He, Liang Yuan, Feifei Chen, Lingjuan Lyu, and Yun Yang. EdgeWatch: Collaborative investigation of data integrity at the edge based on blockchain. In *28th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 2022.
[39] Jingtao Li, Zhezhi He, Adnan Siraj Rakin, Deliang Fan, and Chaitali Chakrabarti. Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing. In *IEEE International Symposium on Hardware Oriented Security and Trust*, pages 248–258. IEEE, 2021.
[40] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation)*, pages 583–598, 2014.
[41] Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. Pipe-sgd: a decentralized pipelined sgd framework for distributed deep net training. In *32nd International Conference on Neural Information Processing Systems*, pages 8056–8067, 2018.
[42] Yuepeng Li, Deze Zeng, Lin Gu, Kun Wang, and Song Guo. On the joint optimization of function assignment and communication scheduling toward performance efficient serverless edge computing. In *IEEE/ACM 30th International Symposium on Quality of Service*, pages 1–9. IEEE, 2022.
[43] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.
[44] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. *ACM SIGARCH Computer Architecture News*, 44(3):255–266, 2016.
[45] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, Ma Yunsheng, Songqing Chen, and Peng Hou. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing*, 11(2):249–261, 2017.
[46] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.
[47] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.
[48] Reiner Ludwig. Who cares about latency in 5g?, 2022.
[49] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. {KungFu}: Making training in distributed machine learning adaptive. In *14th USENIX Symposium on Operating Systems Design and Implementation*, pages 937–954, 2020.

[50] Erfan Farhangi Maleki, Lena Mashayekhy, and Seyed Morteza Nabavinejad. Mobility-aware computation offloading in edge computing using machine learning. *IEEE Transactions on Mobile Computing*, 2021.
[51] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.
[52] Microsoft. Pub/sub messaging, 2022.
[53] Microsoft. What is azure web pubsub service?, 2022.
[54] P Molchanov, S Tyree, T Karras, T Aila, and J Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations*, 2019.
[55] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.
[56] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*, pages 7937–7947. PMLR, 2021.
[57] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on GPU clusters using megatron-LM. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.
[58] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE Symposium on Security and Privacy*, pages 739–753. IEEE, 2019.
[59] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.
[60] Mohanad Odema, Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Lens: Layer distribution enabled neural architecture search in edge-cloud hierarchies. In *58th ACM/IEEE Design Automation Conference (DAC)*, pages 403–408. IEEE, 2021.
[61] Seungeun Oh, Jihong Park, Praneeth Vepakomma, Sihun Baek, Ramesh Raskar, Mehdi Bennis, and Seong-Lyun Kim. Locfedmix-sl: Localize, federate, and mix for improved scalability, convergence, and latency in split learning. In *The ACM Web Conference*, pages 3347–3357, 2022.
[62] Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, 36(10):2333–2345, 2018.
[63] Li Pan, Lin Wang, Shutong Chen, and Fangming Liu. Retention-aware container caching for serverless edge computing. *IEEE INFOCOM Conference on Computer Communications*, 2022.
[64] Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jaesik Choi, Sam H Noh, and Young-ri Choi. HetPipe: Enabling large DNN training on (Whimpy) heterogeneous GPU clusters through integration of pipelined model parallelism and data parallelism. In *USENIX Annual Technical Conference*, pages 307–321, 2020.
[65] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and Mérouane Debbah. Wireless network intelligence at the edge. *Proceedings of the IEEE*, 107(11):2204–2239, 2019.
[66] Junkun Peng, Qing Li, Xiaoteng Ma, Yong Jiang, Yutao Dong, Chuang Hu, and Meng Chen. MagNet: Cooperative edge caching by automatic content congregating. In *The ACM Web Conference*, pages 3280–3288, 2022.
[67] Thomas Rausch, Waldemar Hummer, Vinod Muthusamy, Alexander Rashed, and Schahram Dustdar. Towards a serverless platform for edge {AI}. In *2nd USENIX Workshop on Hot Topics in Edge Computing*, 2019.
[68] Yuxin Ren, Guyue Liu, Vlad Nitu, Wenyuan Shao, Riley Kennedy, Gabriel Parmer, Timothy Wood, and Alain Tchana. {Fine-Grained} isolation for scalable, dynamic, multi-tenant edge clouds. In *USENIX Annual Technical Conference*, pages 927–942, 2020.
[69] Enrique Saurez, Harshit Gupta, Alexandros Daglis, and Umakishore Ramachandran. Oneedge: An efficient control plane for geo-distributed infrastructures. In *ACM Symposium on Cloud Computing*, pages 182–196, 2021.
[70] Yuanming Shi, Kai Yang, Tao Jiang, Jun Zhang, and Khaled B Letaief. Communication-efficient edge ai: algorithms and systems. *IEEE Communications Surveys & Tutorials*, 22(4):2167–2191, 2020.
[71] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321, 2015.
[72] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations*, 2015.
[73] Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 377–390, 2020.
[74] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493, 2022.
[75] Feridun Tütüncüoğlu, Slađana Jošilo, and György Dán. Online learning for rate-adaptive task offloading under latency constraints in serverless edge computing. *IEEE/ACM Transactions on Networking*, 2022.
[76] Lin Wang, Lei Jiao, Ting He, Jun Li, and Max Mühlhäuser. Service entity placement for social virtual reality applications in edge computing. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 468–476. IEEE, 2018.
[77] Lin Wang, Lei Jiao, Jun Li, Julien Gedeon, and Max Mühlhäuser. MOERA: Mobility-agnostic online resource allocation for edge computing. *IEEE Transactions on Mobile Computing*, 18(8):1843–1856, 2018.
[78] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. Dynamic service migration in mobile edge computing based on markov decision process. *IEEE/ACM Transactions on Networking*, 27(3):1272–1288, 2019.
[79] Xiong Wang, Jiancheng Ye, and John CS Lui. Decentralized task offloading in edge computing: a multi-user multi-armed bandit approach. In *IEEE INFOCOM Conference on Computer Communications*, pages 1199–1208. IEEE, 2022.
[80] Zixuan Wang, Joonseop Sim, Euicheol Lim, and Jishen Zhao. Enabling efficient large-scale deep learning training with cache coherent disaggregated memory systems. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 126–140. IEEE, 2022.
[81] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.
[82] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in Neural Information Processing Systems*, 30, 2017.
[83] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: understanding inference at the edge. In *IEEE International Symposium on High Performance Computer Architecture*, pages 331–344. IEEE, 2019.
[84] Xiaoyu Xia, Feifei Chen, Qiang He, John Grundy, Mohamed Abdelrazek, and Hai Jin. Online collaborative data caching in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 32(2):281–294, 2021.
[85] Li Yang, Adnan Siraj Rakin, and Deliang Fan. Rep-net: Efficient on-device learning via feature reprogramming. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12277–12286, 2022.
[86] PengCheng Yang, Xiaoming Zhang, Wenpeng Zhang, Ming Yang, and Hong Wei. Group-based interleaved pipeline parallelism for large-scale dnn training. In *International Conference on Learning Representations*, 2021.
[87] Dixi Yao, Liyao Xiang, Zifan Wang, Jiayu Xu, Chao Li, and Xinbing Wang. Context-aware compilation of dnn training pipelines across edge and cloud. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(4):1–27, 2021.
[88] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. MEST: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34:20838–20850, 2021.
[89] Liang Yuan, Qiang He, Siyu Tan, Bo Li, Jiangshan Yu, Feifei Chen, Hai Jin, and Yun Yang. Coopedge: A decentralized blockchain-based platform for cooperative edge computing. In *The Web Conference*, pages 2245–2257, 2021.
[90] Letian Zhang, Lixing Chen, and Jie Xu. Autodidactic neurosurgeon: Collaborative deep inference for mobile edge intelligence via online learning. In *The Web Conference*, pages 3111–3123, 2021.
[91] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.
[92] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: an extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on computer Vision and Pattern Recognition*, pages 6848–6856, 2018.
[93] Pengpeng Zhao, Anjing Luo, Yanchi Liu, Fuzhen Zhuang, Jiajie Xu, Zhixu Li, Victor S Sheng, and Xiaofang Zhou. Where to go next: A spatio-temporal gated network for next poi recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

A PSEUDOCODE

Algorithm 1: Model Partitioning

Input: t_d: client's probed training time; t_e: worker's probed training time; R: size of training circle; L: number of layers in the test model; $l_\sum$: number of layers in the target model; t_{ul}: uplink transmission time; t_{dl}: downlink transmission time

Output: l_d: number of layers for client

Function Model Partitioning(t_d, t_e, R, L, $l_\sum$, RTT)
- $\overline{t_d} \longleftarrow \frac{t_d}{R \times L}$; /* evaluate client's training performance */
- $\overline{t_e} \longleftarrow \frac{t_e}{R \times L}$; /* evaluate worker's training performance */
- $l_d \longleftarrow (l_\sum * \overline{t_e} + RTT)/(\overline{t_d} + \overline{t_e})$ /* find approximate partition point */
- $l_d \longleftarrow$ round (l_d)
- **return** l_d

Algorithm 2: Partitioning Adaptation

Input: F: training pipeline; l_d: number of layers for client

Output: l_i: partitioning adaptation result

Function Partition Adaptation(F, l_d)
- /* initialize */
- $\forall d \in \{+1, -1\}$ /* set random direction */
- $i \longleftarrow 1, l_i \longleftarrow l_d$
- $F(l_i)$ /* build pipeline */
- obtain t_i from $F(l_i)$ /* obtain training time in c_i */
- /* adapting */
- **while** l_i *not stable* **do**
  - $l_i \longleftarrow l_{i-1} + d$ /* move partition point */
  - $F(l_i)$ with l_i /* adapt pipeline */
  - obtain t_i from $F(l_i)$ /* obtain training time in c_i */
  - **if** $t_i > t_{i-1}$ **then**
    - $d \longleftarrow -d$ /* reverse adaptation direction */
  - **end**
- **end**

Algorithm 3: Worker Adaptation

Input: F: training pipeline; M_t: test model; W: nearby edge servers

Output: w: worker adaptation result

Function Worker Adaptation(F, M_t, W)
- /* initialize */
- probe(W, M_t) /* probe nearby edge servers */
- $w \longleftarrow argmin$(receive(W)) /* find best edge server */
- obtain l_d with Algorithm 2 for w
- $t_{wst} \longleftarrow F(l_d)$
- /* monitor training performance by cycle */
- **For each** *training cycle*
  - $t_{cur} \longleftarrow F(l_d)$ /* obtain training time */
  - **if** $t_{cur} > t_{wst}$ **then**
    - probe(M_t) /* probe nearby edge servers */
    - $w' \longleftarrow argmin$(recv(W)) /* find best worker */
    - **if** $w == w'$ **then**
      - $t_{wst} \longleftarrow t_{cur}$ /* update worst training time */
    - **else**
      - $w \longleftarrow w'$ /* change worker */
      - obtain l_d with Algorithm 2 for w
      - $t_{wst} \longleftarrow F(w, l_d)$
    - **end**
  - **end**
- **end**

return w;

B OPTIMAL PARTITIONING POINT

Fig. 10 shows the optimal partition points for training a VGG-16 across a device-edge pipeline under four different network conditions. We can see that the optimal partition point varies drastically with the network condition.

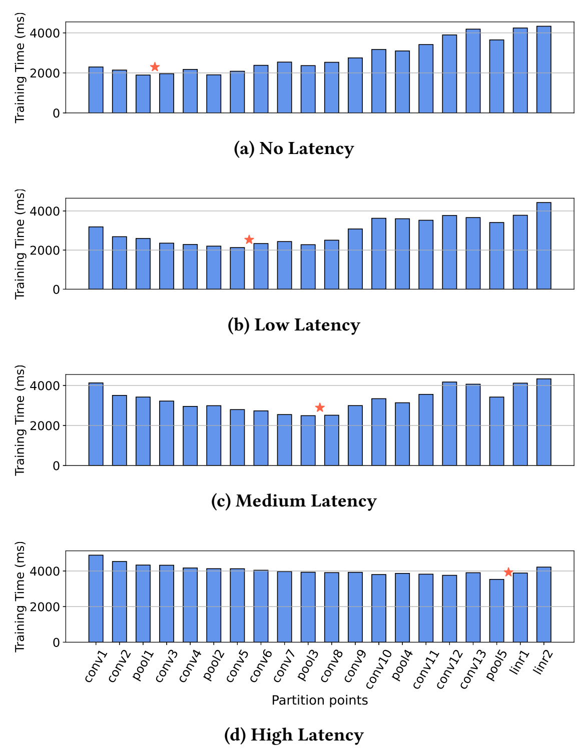


Figure 10: Overall training time for VGG-16 across a device-edge pipeline with different partition points under different network conditions. In this figure, a bar represents the overall training time with the corresponding partition point. The star indicates the optimal partition point that results in the least overall training time.

C EXAMPLE CLIENT TRAJECTORIES

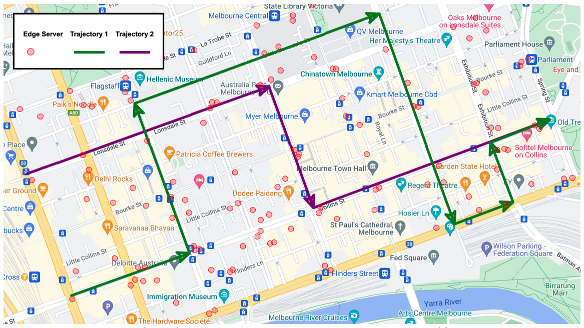


Figure 11: Example client trajectories across Melbourne CBD with 125 edge servers.

D MODELS AND DATASETS

Table 2 summarizes the characteristics of the models and datasets used in the experiments.

Models. VGG-16 [72] and ResNet-50 [23] are conducted on four datasets in the experiments to perform three different ML tasks.

- **Image Classification.** VGG-16 is trained on the CIFAR-10 dataset and ResNet-50 is trained on the CINIC-10 dataset to classify images into 10 classes. The size of the training circles is 500 iterations.
- **Text Classification.** VGG-16 is trained on the AG News dataset to classify articles into 4 classes. The size of the training circles is 1,000.
- **Audio Recognition.** ResNet-50 is trained on the Speech Commands dataset to classify audio records into 12 classes. The size of the training cycles is 200 iterations.

Table 2: Datasets and Models

Dataset	Task	Model	Cycle Size	Classes
CIFAR-10	Image Classification	VGG-16	500	10
CINIC-10	Image Classification	ResNet-50	500	10
AG News	Text Classification	VGG-16	1000	4
Speech Command	Audio Recognition	ResNet-50	200	12

Datasets. Four datasets are used in the experiments.

- CIFAR-10 [16] is the baseline dataset for tiny image classification. It contains 50,000 training image samples and 10,000 test image samples, all sized 32×32. They are equally divided into ten mutually exclusive classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
- CINIC-10 [12] is an augmented dataset for image classification. It is constructed from two popular sources: ImageNet and CIFAR-10. The images are resized to 32×32 and the dataset contains 90,000 images in each its three subsets, including the training subset, the validation subset, and the test subset. It is 4.5 times larger than CIFAR-10.
- AG News [91] is a dataset built from AG's corpus of news articles. It contains 30,000 training articles and 1,900 test articles from the 4 largest classes of AG's Corpus, including "World", "Sports", "Business", and "Sci/Tech".
- Speech Command [81] is an audio dataset of spoken words to help train and evaluate audio recognition systems. It includes 64,727 audio files classified into 12 different classes.