Cartel: A System for Collaborative Transfer Learning at the Edge

Harshit Daga
Georgia Institute of Technology

Patrick K. Nicholson
Nokia Bell Labs

Ada Gavrilovska
Georgia Institute of Technology

Diego Lugones
Nokia Bell Labs

ABSTRACT

As Multi-access Edge Computing (MEC) and 5G technologies evolve, new applications are emerging with unprecedented capacity and real-time requirements. At the core of such applications there is a need for machine learning (ML) to create value from the data at the edge. Current ML systems transfer data from geo-distributed streams to a central datacenter for modeling. The model is then moved to the edge and used for inference or classification. These systems can be ineffective because they introduce significant demand for data movement and model transfer in the critical path of learning. Furthermore, a full model may not be needed at each edge location. An alternative is to train and update the models online at each edge with local data, in isolation from other edges. Still, this approach can worsen the accuracy of models due to reduced data availability, especially in the presence of local data shifts.

In this paper we propose Cartel, a system for collaborative learning in edge clouds, that creates a model-sharing environment in which tailored models at each edge can quickly adapt to changes, and can be as robust and accurate as centralized models. Results show that Cartel adapts to workload changes 4 to 8× faster than isolated learning, and reduces model size, training time and total data transfer by 3×, 5.7× and ~1500×, respectively, when compared to centralized learning.

CCS CONCEPTS

• **Computer systems organization** → *Distributed architectures*; *n-tier architectures*; • **Computing methodologies** → **Machine learning**; *Transfer learning*; *Online learning settings.*

KEYWORDS

Mobile-access Edge Computing (MEC), distributed machine learning, collaborative learning, transfer learning

ACM Reference Format:
Harshit Daga, Patrick K. Nicholson, Ada Gavrilovska, and Diego Lugones. 2019. Cartel: A System for Collaborative Transfer Learning at the Edge. In *ACM Symposium on Cloud Computing (SoCC '19), November 20–23, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3357223.3362708

SoCC '19, November 20–23, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6973-2/19/11…$15.00
https://doi.org/10.1145/3357223.3362708

1 INTRODUCTION

The proliferation of connected devices is causing a compound annual growth rate of 47% in network traffic since 2016, i.e., an increase from 7 to 49 exabytes per month [15]. Service providers such as Facebook, Google, Amazon, and Microsoft rely on machine learning (ML) techniques [18, 47, 50] to extract and monetize insights from this distributed data. The predominant approach to learn from the geographically distributed data is to create a centralized model (see Figure 1a) by running ML algorithms over the raw data, or a preprocessed portion of it, collected from different data streams [12, 37]. More sophisticated solutions deal with geo-distributed data by training models locally in the device, which are later averaged with other user updates in a centralized location – an approach known as federated learning [21, 33, 61, 64].

A centralized model can be very accurate and generic as it incorporates diverse data from multiple streams. From a system perspective, however, there is a challenge in moving all this data, and even the resulting model size can be significant, depending on the implementation, algorithm, and feature set size [2]. Concretely, as data sources spread geographically, the network becomes the bottleneck. In this case, ML algorithms [1, 12, 37], which are efficient in datacenters, can be slowed down by up to 53× when distributed to the network edge [21].

The emergent Multi-access Edge Computing (MEC) architecture, as well as *5G* connectivity, are conceived to converge telecommunications and cloud services at the access network, and have the potential to cope with the challenge described above by enabling unprecedented computing and networking performance closer to users.

In this context, the obvious alternative to centralized learning is to replicate the algorithms at each edge cloud and run them independently with local data, isolated from other edge clouds, as shown in Figure 1b. Isolated models can be useful in certain cases, e.g., when the data patterns observed by an edge are stationary and data is not significantly diverse. However, in more challenging scenarios, where the distribution of input data to the ML model is non-stationary, or when the application requires more complex models – only achievable with more data than the local to a particular edge – isolated models can have prohibitively high error rates (cf. Section 6.2).

Therefore, although MEC and 5G technologies constitute the infrastructure needed to run distributed machine learning algorithms, we argue that there is a need for a coordination system that leverages the edge resources to learn collaboratively from logically similar nodes, reducing training times and excessive model updates.

In this paper we introduce **Cartel**, a new system for collaborative ML at the edge cloud (Figure 1c). The idea behind Cartel is

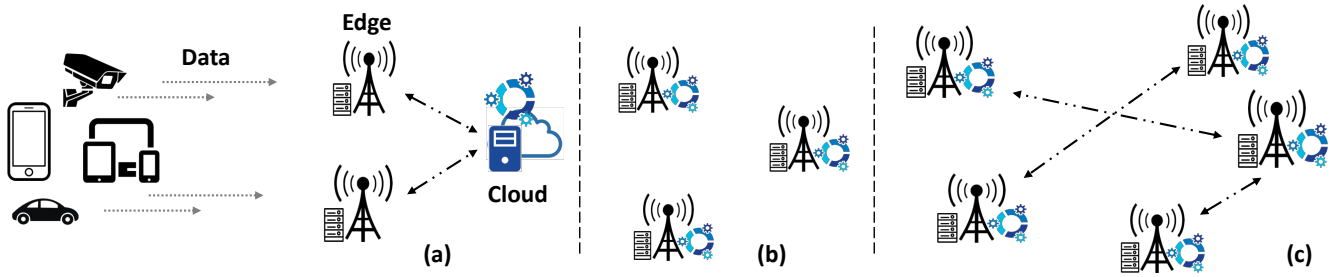


Figure 1: Machine learning systems with geographically distributed data streams, (a) Centralized learning, either raw data or partial models are shipped to a central datacenter for modeling (b) Isolated learning, models are replicated in edge cloud locations and maintained independently, and (c) Collaborative learning (Cartel), a distributed model-sharing framework to aggregate knowledge from related logical neighbors.

that centralized models, although trained on a broader variety of data, may not be required in full at each edge node. When changes in the environment or variations in workload patterns require the model to adapt, Cartel provides a jump start, by transferring knowledge from other edge(s) where similar patterns have been observed. This allows for lightweight models, reduced backhaul data transfer, strictly improved model accuracy compared to learning in isolation, and similar performance to centralized models.

Cartel achieves the above by operating on **metadata**, as opposed to on raw data, and uses metadata to decide *when* an edge-based model needs to be updated, *which peer* should be used for the model update, and *how* the knowledge should be transferred from one model to another. To support these decisions and the collaborative learning they enable, Cartel provides three key mechanisms. It uses *drift detection* to determine variability in the workload distribution observed at the edge (i.e., dataset shift) and in the accuracy of the model. It incorporates functions to identify *logical neighbors*, i.e., candidate edges from which models can be transferred in case of input or output drift. Finally, it supports interfaces with the ML modeling framework to support model-specific *knowledge transfer* operations used to update a model instance in one edge, using state from a model in another edge stack. We describe the system functionality required to support these key mechanisms, their concrete implementation using specific algorithms which operate on model metadata, and evaluate the tradeoffs they afford for different collaborative learning models.

We use both *online random forest* (ORF) [56] and *online support vector machines* (OSVMs) [11, 20] as running examples of learning algorithms. Both ORF and OSVM are well-known online classification techniques [42] that can operate in the streaming setting. Moreover, since they function quite differently, and have different characteristics of the model and update sizes, together they facilitate a discussion of how Cartel can be used with different ML algorithms (see Section 7).

Cartel is evaluated using several streaming datasets as workloads, which consists of randomized request patterns in the form of time series with stationary and non-stationary data distributions at each edge to cover many use cases. We compare Cartel to isolated and centralized approaches. Results show that collaborative learning allows for model updates between 4 to 8× faster than isolated learning, and reduces the data transfer from ~200 to ~1500× compared to a centralized system while achieving similar accuracy. Moreover, at each edge Cartel reduces both the model size, by up to 3×, and the training time, by 3 to 5.7× for the ML models in evaluation.

In summary, our contributions are:

- A collaborative system to create, distribute and update machine learning models across geographically distributed edge clouds. Cartel allows for tailored models to be learned at each edge, but provides for a quick adaptation to unforeseen variations in the statistical properties of the values the model is predicting – e.g., concept drift, changes in class priors, etc.
- Cartel is designed to address the key challenges in learning at the edge (cf. §3) and relies on metadata-based operations to guide cross-edge node collaborations and reduce data transfer requirements in learning (cf. §4).
- We design generic metadata-based operations that underpin the three key mechanisms in Cartel, along with algorithms and system support enabling their implementation. In particular: i) a distance-based heuristic for detecting similarities with other edge clouds, that can serve as logical neighbors for knowledge transfer, and; ii) two generic algorithms for knowledge transfer, one of which (based on bagging) can be applied to any machine learning model. (cf. §5).
- An experimental evaluation using different models, workload patterns, and datasets, illustrates how Cartel supports robust edge-based learning, with significant reductions in data transfer and resource demands compared to traditional learning approaches (cf. §6).

2 MOTIVATION

In this section, we briefly introduce MEC and collaborative learning, and summarize evidence from prior work on opportunities to leverage locality of information in MEC, which motivates the design of Cartel.

Multi-access Edge Computing. Cartel targets the emerging field of MEC [6, 16, 23, 41, 46, 57, 58, 60]. MEC offers computing, storage and networking resources integrated with the edge of the cellular network, such as at base stations, with the goal of improving application performance for the end users and devices. The presence of resources a hop away provides low latency for real time applications, and data offloading and analytics capabilities, while reducing the backhaul network traffic. 5G and novel use cases demanding low latency and/or high data rates, such as connected cars, AR/VR, etc., are among the primary drivers behind MEC [55].

Collaborative Learning. We define collaborative learning to be a model where edge nodes learn independently, but selectively collaborate with logical neighbors by transferring knowledge when needed. Knowledge transfer happens when a *target* edge executing a model detects an issue, such as sudden high error rates, with its current model. Logical neighbors are selected to assist the target edge, that is, edge nodes that: i) are most similar to the target in terms of either the data they have observed or their configuration, and; ii) have models that are performing adequately. Knowledge transfer involves transmitting some part of the model, or models (if there is more than one logical neighbor), from the logical neighbor to the target edge. This framework induces a set of primitive operations that can be directly used with existing ensemble-based machine learning algorithms. Note that the same learning algorithm is used in all edge nodes. For our proof-of-concept, we focus on ORF and OSVM, where the former makes use of bootstrap aggregation (or bagging) [9], but the latter does not. However, we emphasize that since the primitives can be applied to any learning algorithm utilizing bagging, and that since *any* machine learning algorithm can make use of bagging, this means that Cartel is not limited to the two techniques, but rather can be applied in general to any machine learning technique. An interesting future research direction would be to create a general abstraction to apply the set of primitive operations to any machine learning algorithm directly, without the use of bagging. A more surgical approach based on techniques such as patching [30] may be a good candidate for such an abstraction. However, patching raises potential model size issues after repeated application of the primitive operations, and thus further investigation is beyond the scope of this work.

Locality in MEC. As the MEC compute elements are highly dispersed, they operate within contexts which may capture *highly localized information*. For instance, a recent study of 650 base station logs by AT&T [52] reports consistent daily patterns where each base station exhibits unique characteristics over time. Similarly, Cellscope [48] highlights differences in the base station characteristics, and demonstrates the change in a model's performance with changes in data distributions. They also demonstrate the *ineffectiveness of a global model* due to the unique characteristics of each base station. In Deepcham [36], Li et al. demonstrate a deep learning framework that exploits data locality and coordination in near edge clouds to improve object recognition in mobile devices. Similar observations regarding locality in data patterns observed at an edge location are leveraged in other contexts, such as gaming [66] and transportation [3].

3 CHALLENGES

To elaborate on the concepts behind Cartel, we first enumerate the complexities and the key challenges in implementing a distributed collaborative learning system in a general and effective manner. In the subsequent section we introduce our system design, components, and implementation, and explain how each challenge is addressed.

C1: *When to execute the collaborative model transfer among edge clouds?* Since participants run independent models that can evolve differently, each edge must determine when to initiate collaboration with edge peers in the system. Changes in the configuration of the edge stack or in the non-stationary statistical distributions of workloads can decrease model accuracy, thus requiring online retraining. The challenge is to create a mechanism to quickly detect and react to such variations.

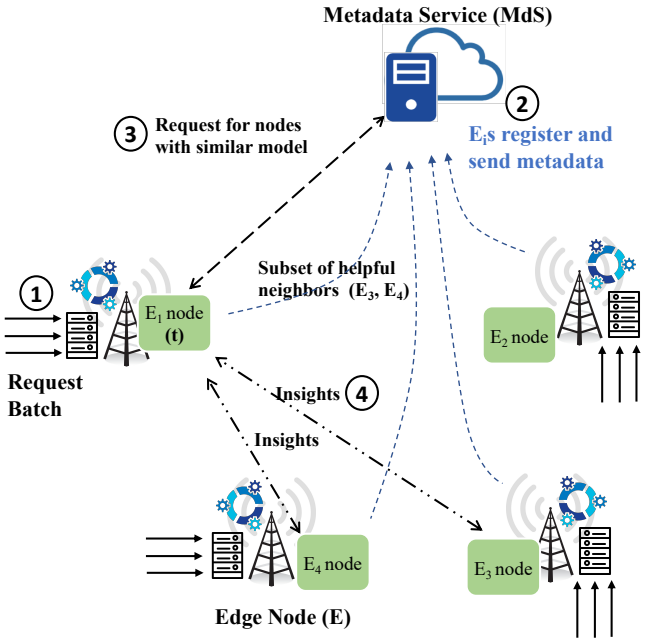


Figure 2: Cartel overview. A collaborative system consisting of edge nodes (E), where E_i's are trained independently and periodically update a MdS with metadata information about the node. A subset of logical neighbors are selected which helps the target edge node (t) to quickly adapt to change.

C2: *Which neighbors to contact?* Our hypothesis for collaborative learning is that edge nodes running similar machine learning tasks can share relevant model portions, thereby achieving more efficient online adaptation to changes. The goal is to avoid sharing of raw data between edge nodes, which makes nodes oblivious of data trends at other edges. Therefore, the challenge is to discover appropriate logical neighbors dynamically while coping with variations in the workload of the multiple edges over time.

C3: *How to transfer knowledge to the target?* In order to update a model it may be possible and/or appropriate to either merge proper model portions from collaborating nodes or to simply replace local models with remote ones. The decision depends on various parameters such as the modeling technique, whether it allows for partitioning and merging of models, as well as the feature set, the pace at which the model needs to be updated, and the efficiency of the cross-node data transfer paths. Thus, the challenge is to provide support for the model-specific methods for sharing and updating model state.

4 OVERVIEW OF CARTEL

Goal. The main focus of Cartel is to reduce the dependence on data movement compared to a purely centralized model that must periodically push out model updates. In contrast, Cartel only performs knowledge transfer when a target node actively requests help from logical neighbors. Thus, when no such requests are active, Cartel only requires nodes to periodically share **metadata**, which is used

to establish a relationship among nodes: raw training data is never explicitly transmitted among the nodes, or to a centralized location.

Concepts. To achieve this goal while addressing the challenges enumerated above, Cartel relies on three key mechanisms: **drift detection** (C1), which allows a node to determine when to send a request to its edge peers for a model transfer; **logical neighbors** (C2), which for each node determines sufficiently similar nodes likely to provide the required model transfer; and **knowledge transfer** (C3), which allows the system to decide on how to merge model portions provided from peer nodes.

A common principle underpinning these mechanisms is the use of system-level support for operating on the metadata. The principle allows Cartel to achieve its desired goal of supporting learning at the edge with adequate accuracy and reduced data transfer costs. As a result, Cartel provides a novel system-level support for metadata management – to store, aggregate, compare, and act on it. This functionality is used by Cartel's key mechanisms, which in turn facilitate collaborative learning.

Metadata can be any information about a node that could potentially distinguish it from other nodes. In other words, information about the physical hardware, software configuration, enabled features, active user distribution by segments, geographic information, estimates of class priors, etc. Some metadata, such as enabled software features, those related to active users, or class prior estimates, can change over time at a given node. When such changes occur, this usually leads to a degradation in model performance, as machine learning techniques are impacted by underlying *dataset shifts*. Examples of such dataset shifts include: changes in class priors (probability of certain classes), covariate shift (distribution from which input examples are sampled), concept shift/drift (classes added/removed or changing boundary functions), and other more general domain shifts (see [45] for a detailed survey).

In our system discussion and experiments, we focus on the first type of shift, i.e., changes in class priors. Thus, in our illustrative online classification examples and experiments, metadata refers specifically to empirical estimates for the prior probabilities for each class, as well as overall and per-class error rates. Such metadata is available in general, and therefore allows us to concretely describe an implementation of each of the three key mechanisms that can be applied in general. We emphasize, however, that additional application specific (or even completely different) choices for metadata are possible, such as the ones enumerated above, to address other dataset shifts.

Architecture Overview. Figure 2 shows the architecture overview of Cartel. The system is comprised of *edge nodes* (E) and a *metadata service* (MdS).At a high level, an edge node maintains a tailored model trained using data observed at the node, and the metadata associated with that model. The metadata service is responsible for aggregating and acting on the metadata generated by edge nodes, so as to facilitate the selection of appropriate peer nodes for collaborative model updates. In other words, when a collaborative model update is requested, the metadata service is responsible for selecting the subset of edge nodes that can share portions of their models, with the node that requests assistance. These edge nodes are then responsible for negotiating with the target to determine which portions to share. Although shown as a single component in the figure, the metadata service is only logically centralized, and may be realized in a distributed manner depending on the scale of the system.

Workflow. The operation of Cartel can be summarized as follows. Each edge node receives *batches* of requests ① over a time period – these requests are the workload on which predictions are made by the *resident model* at the edge node. These batches are later used for retraining the model locally. If an edge node is experiencing poor model accuracy, we refer to that node as the *target* (t). We remark that this batching model fits well with predictive analytics in the streaming setting, e.g., predicting and classifying network resource demands based on the current set of users and their usage patterns.

The metadata from each node is aggregated by the metadata service ② which receives periodic updates from each edge node, described in Section 5.2. Cartel is aimed at scenarios with dynamic workload behaviors. As a result, the neighbor selection cannot be precomputed and stored, but is performed on-demand, based on dynamically aggregated metadata. When an edge node detects that its model accuracy has decreased significantly (drift detection), it asks the metadata service for similar nodes from which model updates should be requested ③. The metadata service processes the metadata on-demand to identify the corresponding logical neighbors. The target node interacts with (one of) its logical neighbors directly to request a model update ④, and applies the shared model state to update its resident model (knowledge transfer).

5 DESIGN DETAIL

Next, we describe the three key mechanisms in Cartel in terms of their metadata requirements and the algorithms they rely on for metadata manipulations, and describe the system support that enables their implementation.

5.1 Metadata Storage and Aggregation

To support its three building blocks, Cartel maintains and aggregates model metadata in the metadata service, and also stores some metadata locally at each edge node. For each of the previous $W \geq 1$ batches, up to the current batch, each edge node maintains: i) counts for each class observed in the batch; ii) the overall model error rate on the batch, and; iii) the error rate per-class for the batch. Here W is user-defined *window length parameter*, which is used to adjust how sensitive the system is to changes in the model metadata. In terms of memory cost, at each edge node Cartel stores $O(CW)$ machine words, if C is the total number of classes in the classification problem.

To aggregate model metadata, the metadata service relies on periodic updates reporting the metadata generated at each edge node. We considered several aggregation policies which trade the data transfer requirements of Cartel for the quality of the selected logical neighbors. The most trivial approach is where edge nodes send updates after every request batch. This helps in ensuring there is no stale information about the edge node at any given time. Thus, $O(CN)$ machine words are transferred after each batch, where C is the number of classes, and N the number of edge nodes. We refer to this operation policy as *regular updates*. These updates can further be sparsified by not sending them for every batch, but

for every m batches, referred to as *interval updates*. Interval updates can provide additional reduction in data transfer, but result in stale data at the MdS. For instance, an edge node may have model performance that degraded recently, but the MdS is oblivious to these changes during logical neighbor selection. One can fix this by adding further validation steps, however, this will delay the collaboration process. An alternative policy is to make use of *threshold updates*, where an edge sends an update only when there is a change in the metadata beyond some user-defined threshold. This reduces the required data transfer, while also attempting to avoid stale information, but requires an additional threshold. In the evaluation of Cartel we primarily use the regular update policy, as it provides an upper bound on how much metadata Cartel must transfer. However, in Section 6.3 we also explore the additional benefits that a threshold update policy can provide.

We remark that Gaia [21] also employs a threshold update methodology. However, a fundamental difference between Cartel and Gaia, is that the later focuses on a new ML synchronization model that improves the communication between the nodes sending the models updates. Though beneficial for models with smaller model update size, the amount of data transfer will increase if applied to models with more memory consumption such as ORF. Further, Gaia's goal is to build a geo-distributed generalized model, whereas Cartel supports tailored model at each node that only seek updates when a change is observed.

5.2 Cartel – Three Key Mechanisms

Drift Detection. As discussed, a dataset shift can cause poor predictive performance of ML models. Through a drift detection mechanism, our goal is to quickly improve the performance of models on nodes where such dataset drift has occurred. Drift detection is a widely studied problem, especially in the area of text mining and natural language processing [62, 63]. It is important to note that in prior work on drift detection, there is often an interest in detecting both positive and negative drifts. However, for Cartel we only take action upon a negative drift (i.e., the error rate of the model increases over time). Any existing drift detection algorithm that can detect negative changes to model error rates can be used in Cartel.

For ease of exposition, we opt for a straightforward threshold-based drift detection mechanism that requires a user-specified *hard limit* $L \in [0, 1]$ on the error-rate of a resident model. Thus, based on the two parameters, L and W, drift detection is performed locally at each edge node after processing a batch. The average error rate of the model is computed on the previous W batches to detect whether the hard threshold L has been exceeded, indicating a drift. Though simplistic, more sophisticated algorithms for drift detection also make use of two (and often more) such thresholds [19]: typically the thresholds are set with respect to statistical tests or entropy measures to determine what constitutes a significant change (cf. [7, 31]).

Logical Neighbor. Although drift detection is useful in determining the *need* for help (i.e., for knowledge transfer) from an external source, we still face the challenge of finding out the node(s) that are most similar to the target in terms of their characteristics: either the data they have observed or their configuration. These nodes are also known as logical neighbors. Intuitively, the goal of Cartel is to find a logical neighbor that has similar class priors to the target node, as this node has most potential to help. Logical neighbors are computed by the metadata service after receiving the request for help from the target node. The mechanism relies on the model metadata collected from each of the edge nodes, and on a similarity measure used to compare models based on the metadata.

Similarity measure. For our example where class priors are undergoing some shift, the empirical distributions from the target node can be compared with those from the other nodes to determine which subset of edge nodes are logical neighbors of the target node. The metadata service maintains a rolling average of this metadata information provided by the edge nodes, which in memory costs can be defined as $O(C\mathcal{N})$ machine words, if C is the total number of classes in the classification problem and $\mathcal{N}$ be the total number of edge nodes in the system. The metadata service is thus responsible for determining which nodes are logical neighbors, and does so via a similarity measure. There are many measures that can be used for this purpose, such as Kullback-Leibler divergence (KLD) [28], Hellinger distance [5], Chi-squared statistic [54] and Kolmogorov-Smirnov distance [44]. After evaluating these techniques empirically, we selected Jensen-Shannon divergence (JSD) [40] (which is based on Kullback-Leibler divergence), as a function to determine the distance of two discrete probability distributions, Q_1 and Q_2: $\text{JSD}(Q_1, Q_2) = (\text{KLD}(Q_1, \tilde{Q}) + \text{KLD}(Q_2, \tilde{Q}))/2$ where, $\tilde{Q} = (Q_1 + Q_2)/2$, and $\text{KLD}(Q, \tilde{Q}) = \sum_i Q(i) \log_2 \frac{Q(i)}{\tilde{Q}(i)}$. JSD has convenient properties of symmetry, and normalized values between $[0, 1]$; this is in contrast with KLD which is unbounded. If $\text{JSD}(Q_1, Q_2) = 0$ then the distributions Q_1 and Q_2 are considered identical by this measure. On the other hand, as $\text{JSD}(Q_1, Q_2) \to 1$ then the distributions are considered highly distant.

Once a list of logical neighbors with high similarity is identified, the list is pruned to only contain neighbors with low model error rates. Neighbors with high error rate, e.g., those that are also currently undergoing dataset drift, are filtered from the list. At this point, the top-k logical neighbors in the list are transferred to the target node, which then negotiates the knowledge transfer. Importantly, if the MdS finds no satisfactory logical neighbors (e.g., if all JSD scores exceed a user-defined threshold), it will return an empty result set to the target node.

Knowledge Transfer. The final step in Cartel is to take advantage of the logical neighbors' models. The knowledge transfer consists of two abstract steps: *partitioning* and *merging*. The knowledge transfer process is dependent upon the machine learning technique used by the application and is accomplished through model transfer – a machine learning technique for transferring knowledge from a source domain (i.e, the data observed by the logical neighbors) to a target domain (i.e., the problematic data arriving at the target node) [49]. The main difference between standard model transfer and this partitioning and merging setting is that there is the potential to transfer knowledge from multiple sources, and also that there is a resident model already present at the target node.

As part of the Cartel system, during the knowledge transfer step, after the target node receives the logical neighbors, the target node attempts to identify those classes that have been most problematic. The target node computes this information by examining the per-class error rates over the last W batches. Any classes that have

error-rates exceeding a user defined threshold are marked as problematic, and communicated to the logical neighbors as part of the request for help. Next, depending on the machine learning algorithm running on the edge nodes, partitioning and merging proceeds in different ways. We give two different methods for partitioning and merging that can be applied broadly to any classification problem.

Bagging Approach. For the case of ORF, or any online learning algorithm that uses bootstrap aggregation, knowledge transfer is achieved via the following straightforward technique. Suppose the model at the target node contains $\mathcal{M}$ sub-models, each constructed on a separate bag: e.g., in the case of ORF, $\mathcal{M}$ is the number of online decision trees in the forest. By replacing a $\mathcal{Z} \in (0, 1]$ fraction of the sub-models in the target ensemble ($\mathcal{Z}$ of the trees from the target in the case of ORF), with sub-models collected from the logical neighbors, we can intuitively create a hybrid model at the target node, that is somewhere in between the old model and the models from the logical neighbors. In other words, we partition the models in the logical neighbors, selecting $\mathcal{Z} \times \mathcal{M}$ sub-models among the logical neighbors, and then merge these with the existing model at the target node.

Before discussing how to set $\mathcal{Z}$, it first makes sense to answer the question of which trees should be replaced in the target ensemble, and which trees should be used among those at the logical neighbors. We employ the following heuristic: replace $\mathcal{Z}$ of the trees having the highest error rate in the target node ORF, with the $\mathcal{Z}$ trees having the lowest error rate from the logical neighbors. To achieve partitioning and merging with bagging, Cartel must therefore additionally maintain, at each edge node, the error rates of each sub-model (e.g., decision tree in the forest). Fortunately, for many libraries implementing ensemble ML algorithms (such as scikit-learn [51]), this information is readily accessible from the model APIs. We also experiment with another heuristic that replaces $\mathcal{Z}$ trees with the highest error rate in the target node ORF, with the $\mathcal{Z}$ trees that have the lowest per-class error rate among the problematic classes.

The exact setting of $\mathcal{Z}$, in general, can depend on workload dynamics, as well as the distance between the target node and logical neighbors. For our datasets and workloads we experimentally found that $\mathcal{Z} = 0.3$ worked well, and discuss this later in Section 6.3. However, other choices, such as $\mathcal{Z}$ in the range $[0.3, 0.6]$ also behaved similarly. Thus, a precisely engineered value of $\mathcal{Z}$ is not required to yield similar benefits for the workloads and datasets we used. We leave automatic online tuning of $\mathcal{Z}$ as an interesting topic for future investigation.

One-versus-Rest Approach. In contrast to the bagging approach above, linear OSVMs using a one-versus-rest (or one-versus-all) approach to multi-class classification to construct a set of C hyperplanes in an n-dimensional space, one for each class, each defining boundary between its associated class and items not in that class. This boundary is therefore represented as a row in a $C \times (\mathcal{F} + 1)$ weight matrix containing the coefficients – each associated with one feature plus an additional bias term – representing the hyperplane, where $\mathcal{F}$ is the number of features for the model. For OSVM, knowledge transfer can be accomplished by updating the weights assigned to the features of problematic classes. The logical neighbors then partition the subset of these problematic classes from their weight matrices, by simply selecting the rows corresponding to these classes. These rows are transmitted to the target node where the node merges these model portions into its OSVM model by overwriting the corresponding rows with the weights from the logical neighbors. We note that the same approach applies in general to other one-versus-rest classifiers, but that it is especially appealing in the case of linear models like OSVM, as the data required to transmit the $\mathcal{F} + 1$ weights is small compared to other approaches.

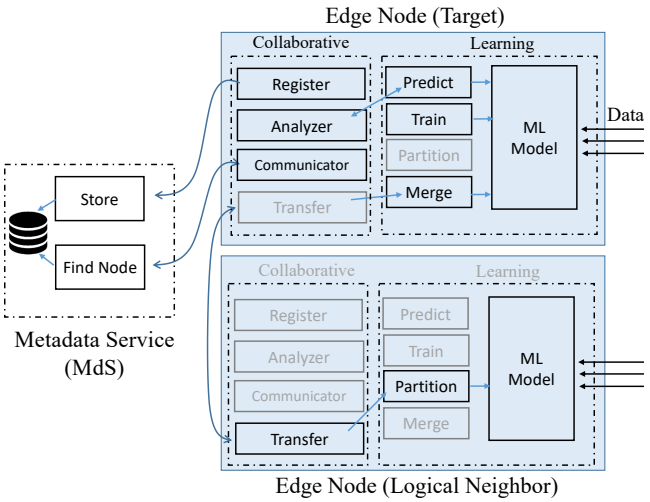


Figure 3: Cartel system functions.

Crucially, both methods presented have the property that the resulting model at the target node does not increase in size. In particular, the same number of sub-models is present for the bagging approach and for OSVM the matrix representing the hyperplanes is exactly the same size in terms of the total number of entries. This means that knowledge transfer can be repeated without gradually inflating the size of the model.

The focus for Cartel is to provide the system support for partitioning and merging operations to be easily integrated for collaborative learning. The specific implementation of partitioning and merging is left to the user, beyond the generic approaches just described. Thus, Cartel abstracts this mechanism as a set of APIs for partitioning and merging that can be extended by any machine learning model to be incorporated into the Cartel system, as shown in Figure 3 and described in the next section.

5.3 Cartel Runtime

The Cartel runtime at each edge node consists of two blocks of functions – Learning and Collaborative – as shown in Figure 3. The *Learning component* depends on the machine learning technique used and the type of problem it addresses (e.g., classification, regression, etc.). It constitutes the learning part of the system which makes predictions on the incoming data using the model, compares the predicted values to later observations determining the error rate, and subsequently re-trains the model using observations. It provides four interfaces: **predict**, **(re)train**, **partition**, and **merge**. The *predict* function keeps track of overall and class-wise results predicted by the model while the *(re)train* function is responsible for training the model on the incoming data feed.

Moreover, when a model update is requested by a target node, the *partition* function of the logical neighbor helps in finding the portion of the model that increases the model's accuracy at the target edge node. Finally, the *merge* function of the target node incorporates the model update received from the supporting edges and completes a cycle in a collaborative learning process.

The *Collaborative component* is independent of the machine learning technique used. It is responsible for drift detection, for triggering look-ups and for interacting with logical neighbors. It provides four functions – **register**, **analyzer**, **communicator**, and **transfer**. With *register* an edge node joins the Cartel knowledge sharing pool. The *analyzer* function analyzes the prediction results to determine a drift. Upon drift detection, it additionally performs data trend analysis to identify the problematic classes at the target node. The *communicator* function interacts with MdS to update the node's metadata information, based on the metadata aggregation policy. Additionally, if a drift is detected, it also sends a request to MdS for logical neighbors. The *transfer* function opens a communication channel between the target node and the selected logical neighbor(s) to request and receive model portions.

6 EVALUATION

We present the results from the experimental evaluation of Cartel, and seeks to answer the following questions:

1. How effective is Cartel in reducing data transfer costs, while providing for more lightweight and accurate models that can quickly adapt to changes? (Section 6.2);
2. What are the costs in the mechanisms of Cartel and the design choices? (Section 6.3);
3. How does Cartel perform with realistic applications? (Section 6.4).

6.1 Experimental Methodology

Experimental Testbed. We evaluate Cartel on a testbed representing a distributed deployment of edge infrastructure, with emulated clients generating data to nearby edge nodes. The testbed consists of five edge nodes and a central node representing a centralized datacenter. All nodes in the system are Intel(R) Xeon(R) (X5650) with 2 hex-core 2.67GHz CPUs and 48GB RAM.

Datasets and applications. The first experiment uses an image classification application where edge nodes participate in classifying images into different categories using ORF and OSVM. The models are implemented using the ORF library provided by Amir el at. [56] and, for OSVM, scikit-learn [51]. We use the MNIST database of handwritten digits [35], that consists of 70k digit images. A set of 1000 uniformly randomly selected images (training data), distributed across each of the edge node, is used for preliminary model training. The remainder of the dataset is used to generate a series of request patterns, following the different distribution patterns described below; batches from these requests are used for online training.

We also evaluate Cartel with a second use case based on network monitoring and intrusion detection (Section 6.4) that uses the CICIDS2017 Intrusion Detection evaluation dataset [59]. This use case further illustrates some of the tradeoffs enabled by Cartel, and helps generalize the evaluation. The CICIDS2017 dataset consists of a time series of different network measurements, preprocessed into feature values, and includes a mix of benign data samples as well as malicious attacks such as distributed denial-of-service (DDoS) attacks, Heartbleed, web and infiltration attacks captured over a period of 5 days. We use a subset of the features for two days of the dataset consisting of benign data samples, DDoS attacks, and port scan attacks, consisting of 500k data samples in total.

Workloads. Edge nodes process requests in discrete batches over time. A batch consists of varying number of requests corresponding to different classes in a dataset (e.g., corresponding to one of the ten different digits from 0 to 9 in the case of MNIST, or to a different type of attack in case of CICIDS2017). In our experiments, the dataset shift we focus on is a change in class priors, i.e., a change in the distribution of the classes arriving at a node.

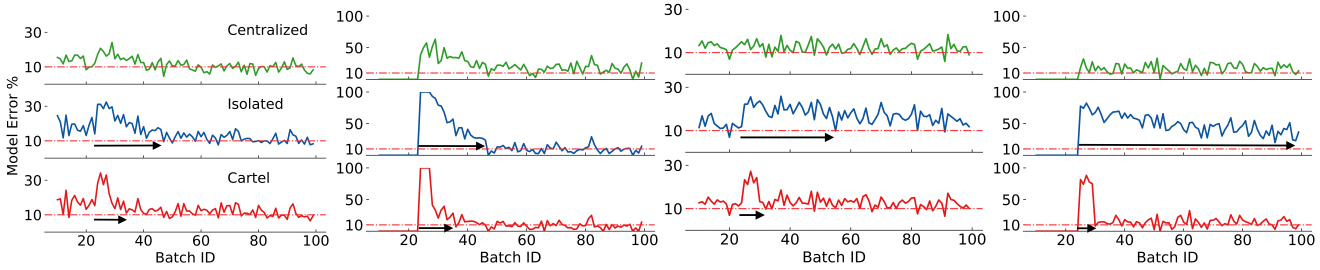
We generate several synthetic request patterns which correspond to different types of change patterns in the workload. The results presented in the paper are primarily based on the *Introduction* and *Fluctuation* patterns. *Introduction* corresponds to a case where a new class gets introduced at an edge node abruptly after 25 batches. *Fluctuation* is a distribution pattern where a new class is introduced at batch 25, but then disappears and re-appears at batches 50 and 75 respectively. This is analogous to the *Introduction* pattern with periodicity. Other patterns used in our evaluation include *Uniform*, where all classes are uniformly distributed across all edge nodes, and *Spikes*, where several new classes are introduced in succession, each of which does not persist. The results obtained from these latter two workloads are similar, so we omit them for brevity.

We have used emulated nodes and synthetic workloads, primarily because of limited availability of real infrastructure and data. However, in the following section we successfully demonstrate the benefits of Cartel. Moreover, with more edge nodes, we expect the savings from transmitting metadata compared to the raw data will persist.

6.2 Benefits from Cartel

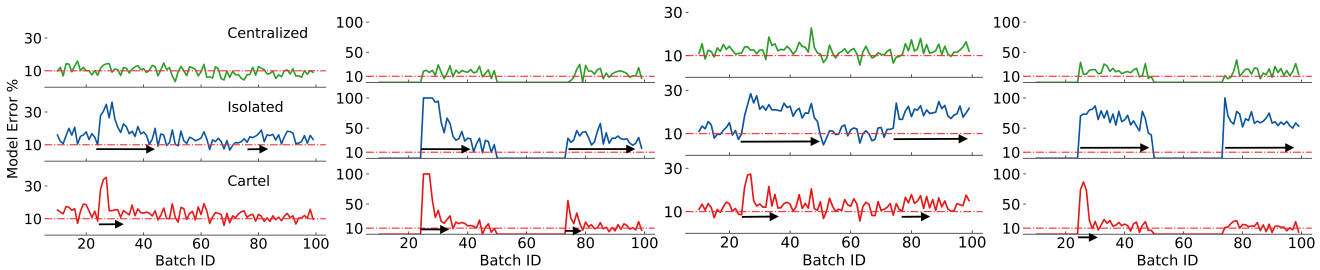
We compare Cartel to centralized and isolated learning, with respect to the changes observed at the edge in the three systems. A centralized system repeatedly builds a generic model using data collected from all the nodes in the system. This model is then distributed among the edge nodes. In such a system there exists a gap between the error bound and the model performance at the edge node. This is due to the time difference between the periodical update of the model at the edge nodes. In contrast, in an isolated environment, each edge node is trained individually and any change(s) in the workload pattern could impact the predictive performance of the model.

We measure the time taken to adapt to changes in the class priors in the workload, and examine the resource demand (cost) in terms of data transferred over the backhaul network, time required to train the online model and model size. In Figures 4 and 5, we present the results for the image classification application with ORF and OSVM, and the *Introduction* and *Fluctuation* patterns. We use different workload patterns to assess the impact of change in request distribution on the performance of the systems. For each case, with a horizontal dashed red line, we show the error lower bound, obtained with offline model training. We used window size



(a) ORF overall error **(b)** ORF class error **(c)** OSVM overall error **(d)** OSVM class error

Figure 4: Performance comparison for introduction workload distribution (lower error rate is better) showcasing overall model error and the introductory class error change. Misclassification of introductory class degrades the model performance. Cartel is able to quickly adapt to the change in distribution (given by the horizontal arrow). The horizontal line (in red) defines the lower bound obtained through an offline training.



(a) ORF overall error **(b)** ORF class error **(c)** OSVM overall error **(d)** OSVM class error

Figure 5: Model performance comparison for fluctuation workload. Similar to Figure 4 Cartel is able to adapt to the changes in the distribution and behaves close to the centralized system.

$W = 5$ and hard error rate limit of $L = 0.15$ for Cartel and $W = 10$ for the centralized system to provide the model updates at the edge.

Adaptability to Change in the Workload. We observe that the centralized system is more resilient to the change in the distribution pattern. This is due to the generic nature of the edge model which is regularly synchronized with the central node and is built using prior knowledge from the other edge nodes. On the other hand, the isolated system and Cartel experience a spike in the model error rate for the same change. We define the time taken for the model error rate to return to the baseline (10%) as the adaptability of the system to change. This adaptability is denoted with a horizontal arrow in Figures 4 and 5.

Cartel's drift detection allows the target node to have increased adaptability with respect to the dataset shift (measured as a smaller horizontal spread in terms of number of batches) when compared to an isolated system. Specifically, when using OSVM and ORF techniques, Cartel performs 8× and 4× faster, respectively, as compared to an isolated system. The adaptability of Cartel is important for both workload patterns. For *Fluctuation* (Figure 5), Cartel helps to bring the system back to an acceptable predictive performance while the isolated system takes a longer time to adapt to the fluctuation.

Data Transfer Cost. In a centralized system, an edge node proactively updates its model from the central server which helps in improving the inference at edge nodes. However, this improvement comes at a cost of a proactive model transfer between the edge and the central node. To capture the network backhaul usage we divide the data transferred into two categories: (i) data / communication cost which includes the transfer of raw data or metadata updates, and (ii) model transfer cost which captures the amount of data transferred during model updates to the edge (periodically in case of a centralized system or a partial model request from a logical neighbor in Cartel).

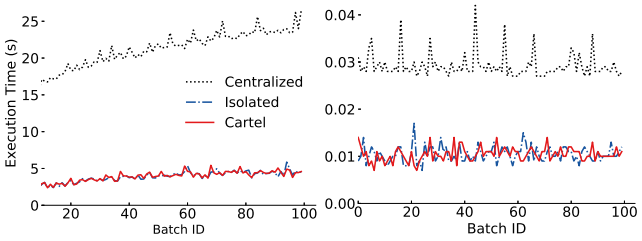
Cartel does not centralize raw data and only transfers models when there is a shift in the predictive performance. This design helps in reducing both data / communication cost and model transfer cost by ~1700×, and 66 to 200×, respectively, thereby reducing the overall cost of total data transferred for Cartel by two to three orders of magnitude, compared to the centralized system, as shown in Table 1.

We note that for ORF the cost of the model updates is the dominating factor in the total data transferred, whereas the data / communication dominates for OSVM. As discussed, the data / communication for Cartel is $O(CN)$ per batch. For a centralized model, the data / communication is $O(BFN)$ where B is the average number of data points in a batch. Provided $BF \gg C$, we can expect the data / communication cost to be much lower for Cartel than a centralized system.

For applications where dataset shifts are less frequent, we expect Cartel will provide better predictive performance in the long run. We expect the gains of Cartel to persist even when considering federated learning-based approaches to building a centralized model [33]. Those are reported to reduce communication costs by one to two orders of magnitude, but, importantly, they strive to build at each edge a global model, and can miss the opportunity

ML	Workload	D/C (×)	MU (×)	Total (×)
ORF	Introduction	1745	200	212
	Fluctuation	1760	66	71
OSVM	Introduction	1763	188	1573
	Fluctuation	1763	94	1404

Table 1: Ratio of data transferred in a centralized system versus Cartel. D/C represents data/communication, MU represents model update transfer cost and Total represents the combined cost.



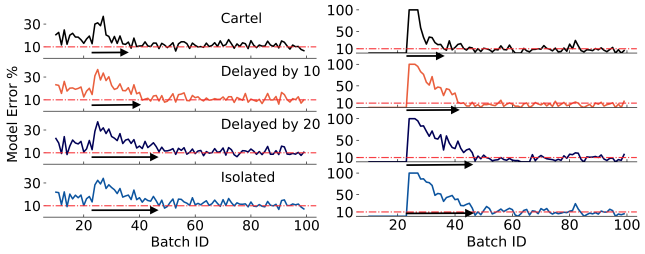
(a) ORF **(b)** OSVM

Figure 6: Time required to train the ORF and SVM model. Similar trends are observed for different workload distribution. Combination of global model and bigger training dataset in a centralized system increases the online training time of ML model.

to benefit from reduced model sizes or training times, as discussed next.

Model Size. The model size depends on the machine learning technique used. It plays an important role in data transfer during model updates as mentioned above, as well as during retraining of the model. Cartel results in smaller, tailored models for each edge node, leading to faster online training time. Since ORF is an ensemble learning method, it builds multiple decision trees, each with varying number of nodes. The size of the ORF model depends on several factors, such as the data and hyperparameters used. From our experiments with two edge nodes, we observe that Cartel results in a reduction of the model size by 3× on an average when compared to a centralized system. This reduction is achieved because a tailored model in Cartel does not store as much information when compared to a generic model used in a centralized system. This is expected in MEC, because it operates in contexts with highly localized information that can result in fewer classes being active or observed at each edge [48, 52]. Beyond reduction in classes, the number of nodes in the ORF grows less quickly in Cartel vs. the centralized system due to fewer total training examples, further reducing the model size. For ORF, the model resulting from use of Cartel is similar to that of isolated learning, but has faster adaptability (as shown in the above discussion). Since OSVM uses a matrix to represent the hyperplane parameters corresponding to each class, there is no difference (without applying further compression) in size of an OSVM model trained for subset of classes compared to one trained using all classes.

Training Time. The online training time for a machine learning model is a function of the training dataset and the model size. In an online system, a smaller model size and/or less data helps in training the model faster. Figure 6 shows the difference in the training times for the ORF and OSVM model during our experiment. The smaller model size and smaller local batch size reduces the ORF training time up to by 5.7×, while for OSVM, since the model size is constant, the smaller training batch size alone reduces the training time by 3× compared to a centralized system.



(a) Overall model error **(b)** Introductory class error

Figure 7: Effect of different drift detection policies on overall model and introductory class error rate: "Delayed by X" implies the drift detection was delayed by X batches.

Impact of Machine Learning Algorithm. We remark that benefits from Cartel are not primarily related to the ML model accuracy, but rather to its adaptability during distribution changes. In the case of OSVM, although linear SVM exhibits high bias on the MNIST data, adding more features or using kernel SVMs are unlikely to speed up convergence. As such we expect the benefits of Cartel to persist even when other non-linear methods or feature transformations are used. To test this, we ran the MNIST workload as before, but with Random Fourier Features (RFF) [13, 53] to improve model accuracy.[1] Although these additional features lowered the average error rate, we observed similar improvements to adaptability, as well as a reduction in data transfer, as observed with linear SVM and ORF. Thus, Cartel can provide similar benefits when used with these additional techniques.

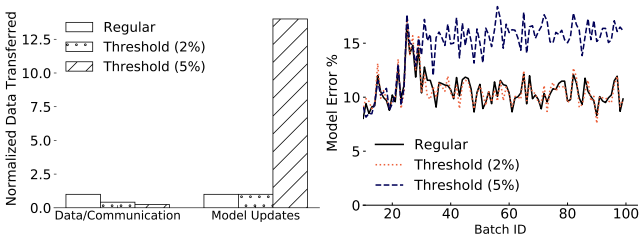
In summary, Cartel boosts the system's adaptability by up to 8×. It achieves a similar predictive performance compared to a centralized system while reducing the data transfer cost up to three orders of magnitude. Cartel enables the use of smaller models at each edge and faster training.

6.3 Effect of Mechanisms

We next investigate the impact that each of the mechanism in Cartel has on the overall system performance.

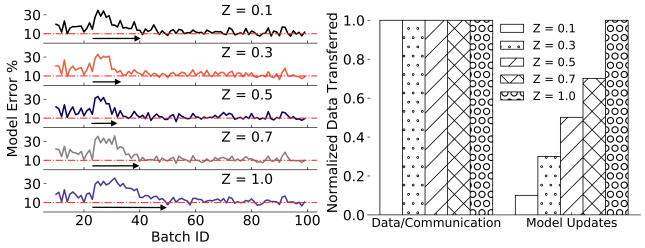
Drift Detection Timeliness. Timely drift detection is important for Cartel, since a delay in detection can impact a model's predictive performance. To demonstrate the impact of slow drift detection, we modify the drift policy to delay the request to the MdS for logical neighbors by a variable number of batches. The results in Figure 7, show the impact of drift detection delay on the overall model's performance as well as the misclassification rate of the introduced class for ORF; we observe a similar pattern for OSVM. The request for model transfer from a logical neighbor at the actual time of drift detection stabilizes the system quickly. If the delay is too large (e.g., in the figure, a delay of 20 batches), the model transfer does not provide collaboration benefits, as online training eventually improves the model. In the current implementation of Cartel, drift detection triggers immediate requests for logical neighbors. We acknowledge that overly sensitive drift detection may cause short and non-persistent workload fluctuations to trigger

[1] We used the implementation of Random Fourier Features of Ishikawa [25].



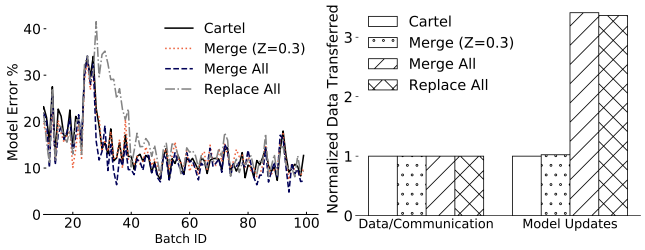
(a) Data transfer **(b)** Overall model error

Figure 8: Comparison of MdS metadata aggregation policies.



(a) Overall model error **(b)** Data transfer

Figure 9: Comparison of adaptability of Cartel and total data transfer for various Z values when using ORF ML model on MNIST dataset.



(a) ORF overall model error **(b)** ORF data transfer

Figure 10: Comparison of adaptability of the system and total data transfer between various knowledge transfer mechanisms, on MNIST dataset using ORF as machine learning model, that can be applied in Cartel.

unwarranted and frequent transfer requests, increasing the overall data transferred over the network; detailed sensitivity analysis can be used to develop automated methods for determining effective ranges for the delay parameter in the future.

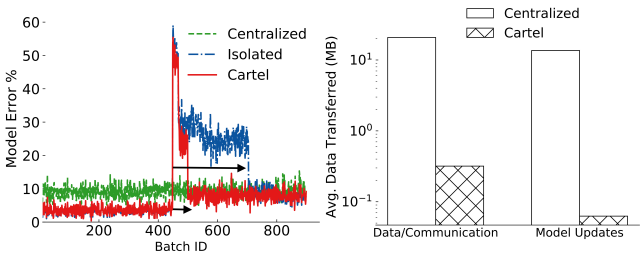
MdS Aggregation Policies. The metadata at MdS can be updated according to different policies described in Section 5.1. All other experiments use the "regular" update policy, but we note that additional data reduction benefits can be obtained through the "threshold" update policy when configured appropriately. Figure 8a shows a comparison of the regular update policy with different threshold policies. Here we use a 2% and 5% change in class priors at each edge as the threshold parameter. As the threshold increases, we observe a reduction in the total metadata transfer (labeled as data/communication cost transfers in the figure). However, too high thresholds could result in an increase in model update costs. Use of a threshold parameter can result in a misrepresentation of the distribution pattern of different edge nodes at the MdS. The corollary to this is the selection of incorrect logical neighbors, shown in Figure 8b, where the system repeatedly requests for model updates and fails to adapt to the changes in the system due to incorrect logical neighbors, negating the benefit of reducing other communication costs.

Finding Logical Neighbors. In a distributed deployment with hundreds of edge nodes, collaboration can be performed by: i) selecting a node at random, ii) based on geographical proximity, or; iii) based on similarities determined through node metadata information, for example. We experiment with the impact of Cartel's logical neighbor selection, compared to various baseline approaches, using the *Introduction* workload and ORF. The logical neighbor algorithm in MdS is modified to introduce i selection failures (by randomly selecting among the nodes not identified as a top match by the MdS algorithm), before finally providing the correct logical neighbor. Depending on the batch size and how long drift detection takes, multiple failures – in our experimental setup, more than two – negate any benefit that knowledge transfer has on the accuracy of the model. Thus, it is critical that the MdS uses timely metadata about edge nodes and effective similarity measure, to identify good logical neighbor candidates.

Knowledge Transfer Balance. As discussed in Section 5.2, knowledge transfer for OSVM involves selecting the coefficients associated with the hyperplane for the problematic classes. However, ORF operates by selecting the $\mathcal{Z}$ trees with lowest error rate for partitioning at a logical neighbor, raising the question of the value of $\mathcal{Z}$. For our experimental setup, we tested the following values of $\mathcal{Z}$: $0.1, 0.2, \ldots, 1.0$. We found that $\mathcal{Z} < 0.3$ fails to help for the problematic class, while $\mathcal{Z} > 0.6$ leads to higher error rate for the non-problematic classes at the target node, as shown in Figure 9. Both result in more time required by Cartel to adapt to the changes. Hence, we selected $\mathcal{Z} = 0.3$ as the optimal value, since higher $\mathcal{Z}$ values, for ORF, result in more bandwidth utilization.

In addition, there are a few ways to apply a model update. In Figure 10 we evaluate the impact of each of these on the performance of Cartel, in terms of its ability to quickly converge to good models at the target node (with low model error rate) (left hand-side graphs in the figure), and in terms of the data transfer requirements (right hand-side graphs). For ORF, one can replace the existing forest with the logical neighbor's forest (Replace All); two or more forests can be merged (taking the union of all the trees) (Merge All); the best performing $\mathcal{Z}$ of trees from the logical neighbor can be merged with the target's forest (Merge ($\mathcal{Z} = 0.3$)); or, finally, the worst performing trees in the target model can be replaced by the best performing $\mathcal{Z}$ trees from the logical neighbor's model. The latter is what is used in Cartel, and enabled by use of additional local metadata at each edge, examined during the knowledge transfer request.

Replacing the entire edge model with the neighbor's model might not work because each edge node experiences different distributions and a blind merge from a logical neighbor would not work if only a few classes were common among the nodes. When possible (i.e., for ORF), merging all or a portion of the model (the ORF trees) seems to be a good solution when considering the error convergence time at the target. However, this increases the overall model size by up to 2× for Merge All, which further results in increase in training time by an average of 2× for Merge All, or 1.3× for Merge ($\mathcal{Z} = 0.3$). Replacing portions of the target model based



(a) ORF overall error **(b)** Data transfer

Figure 11: Performance and total data transfer comparison for network attack dataset using ORF to classify begin request against DDoS and port scan attack requests.

on the top performing $\mathcal{Z}$ classes in the neighbor's model results in a target model that quickly converges to the model's lower error bound, and keeps model transfer costs low. By enabling replacement of only those model portions which relate to the problematic classes at the target, Cartel achieves both quick adaptation and low transfer costs.

6.4 Use Case - Network Attack

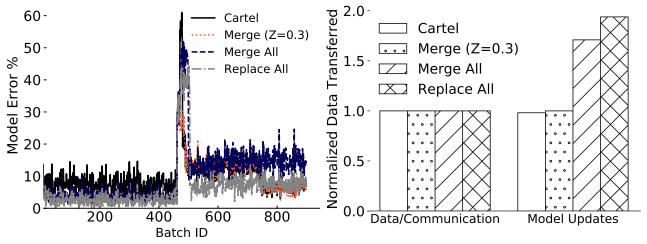
This scenario is based on data from the CICIDS2017 Intrusion Detection evaluation dataset [59]. The testbed consists of two edge and a central node. One of the nodes ($Edge_0$) experiences a sustained port scan attack, while the other ($Edge_1$) experiences a DDoS attack. After a time period, $Edge_0$ (target node) experiences a similar DDoS attack. The ML model used to classify the attacks is ORF, with 10 tree predictors at each node, each with a maximum depth of 16. The workload follows the *Introduction* distribution and consists of 900 batches each with ~1000 data points of various network metrics (features).

Result. As shown in Figure 11, the collaboration with a logical neighbors helps the target node to adapt 5× faster to the introduction of the DDoS attack which was already observed by $Edge_1$, compared to the isolated system. A centralized system with edge nodes receiving regular model updates from a central node does not require time to adapt, however, even with only two edge nodes, the total data transfer is 90× more, and the time taken for training is 2× that of Cartel.

We performed a more comprehensive evaluation of Cartel using the intrusion detection dataset. Figure 12 demonstrates that the knowledge transfer mechanism in Cartel reduces model transfer cost 2.5× or more compared to the other mechanisms, while keeping a lower overall model error rate. Additionally, the results from the *Fluctuation* workload exhibit increased adaptability of the system, reduced total data transfer (by 60×) and faster model training time (by 1.65×), compared to centralized learning. These results showcase a similar trend to the ones described for the MNIST-based use case in the Section 6.2; the graphs are omitted for the brevity.

7 DISCUSSION

We have shown how Cartel performs, in terms of data transfer, training time and model size. Our results demonstrate the potential of Cartel, but also illustrate several opportunities to be further explored.



(a) ORF overall model error **(b)** ORF data transfer

Figure 12: Comparison of three knowledge transfer mechanisms applied to the network attack dataset using ORF as the classification model.

Collaboration Scope. We evaluate Cartel under different data distribution scenarios. However, the evaluation is performed using synthetic workloads in an attempt to model realistic scenarios [52], and is limited to few edge nodes. Given that a centralized model must periodically be pushed to all edge nodes, we expect that the data transfer reduction of Cartel will be larger in deployments with dozens or hundreds of edge nodes. Thus, it would be interesting to evaluate the benefits of Cartel in such scenarios, and for use cases with live traffic and other dataset shifts. In particular, the choice of some of the parameters and threshold values chosen in Cartel, is dependent upon the workload and the characteristics of the underlying infrastructure, and further work is needed to establish practical policies for choosing effective values.

Generalize to other machine learning algorithms. We present the benefits of Cartel with the underlying machine learning algorithms as ORF and OSVM, and developed general partitioning and merging algorithms that work in the bagging and one-versus-rest paradigm. Still, we plan to explore other methods for partitioning and merging heuristics that can be used directly (rather than requiring bagging). We are interested in general methods for deep neural networks (DNN), and also in evaluating online regression problems in addition to online classification. As mentioned, one possibility is patching [30], though further developments are needed to ensure models do not become excessively large over many partitioning and merging operations. Recent work on knowledge transfer through merging of DNN [4, 14, 67] could be a stepping stone in extending Cartel to support DNN models. Other recent work has been done to partition DNNs across mobile, edge and cloud [22, 26, 29, 32], yet additional advances are needed in the ML algorithms to improve their efficacy of model transfer.

Privacy. While a discussion about privacy is beyond the scope of this paper, we note that edge nodes in Cartel use the raw data to train models, but do not explicitly transmit this data to the other nodes. As such, the information sent to the MdS or to logical neighbors is a sketch derived from the raw data, e.g., a histogram of the data distribution at a node. However, we still foresee concerns about this sketched data, and believe such concerns also apply to other techniques such as federated learning. Regarding the possibility and handling of malicious edge nodes, our scope is limited to cooperating nodes that are owned or managed by a single service provider. We leave further exploration of issues, such as trust, as future work.

8 RELATED WORK

Cartel is a system that leverages the unique characteristics of each edge and enables collaboration across nodes to provide a head start in adapting the model for changes observed at the target node. This is in contrast to the existing systems [38, 43, 65] where data processing and analysis happens in a single datacenter, however, the excessive communication overhead in distributed machine learning algorithm makes such systems unsuitable in a geo-distributed setting.

Systems such as Gaia [21], Project Adam [12], Federated learning [33] and others [10, 39, 64] focus on addressing the communication overhead in running machine learning methods such as a parameter server and large deep neural network in a geo-distributed environment. Additionally, the distributed setting involves interaction with a large number of nodes, where some of these nodes can experience failures. MOCHA [61] is a system designed to handle such stragglers. DeepCham [36], IONN [26], Neurosurgeon [29] and Splitnet [32] are examples of systems where the machine learning model is partitioned across mobile, edge or cloud which works in a collaborative way to train a unified model. These systems do not consider custom models for each node in MEC where an edge might not require a global model trained on broad variety of data.

Similarly to Cartel, Cellscope [48] is also aimed at creating better models at edge nodes. Using real data, the authors show evidence that global models can lead to poor accuracy and high variance. However, the focus of that work is on providing a bigger dataset by intelligently combining data from multiple base stations to help in building the local model at edge nodes. In contrast, Cartel avoids data transfer and aims to provide model updates from logical edge nodes only when there exists a data shift.

Finally, there exist many machine learning algorithms [8, 24, 27, 34] to incrementally train machine learning models in an efficient manner and more sophisticated knowledge transfer techniques [17, 49] that Cartel can leverage to further improve the learning performance.

9 CONCLUSION

In this paper we introduce Cartel, a system for sharing customized machine learning models between edge datacenters. Cartel incorporates mechanisms to detect changes in the input patterns of the local machine learning model running in a given edge, to dynamically discover logical neighbors that have seen similar trends, and to request from them knowledge transfer. This creates a collaborative environment to learn from other models, only when required, and without sharing the raw data. Experiments show that Cartel allows edge nodes to benefit from the use of tailored models, while adapting quickly to change in their workloads, and incurring significant reductions in data transfer costs compared to approaches based on global models. As future work, we aim to explore the opportunities for additional gains from algorithmic improvements while adding other machine learning models to the system.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Amazon. 2018. Amazon Machine Learning. System Limits. https://docs.aws.amazon.com/machine-learning/latest/dg/system-limits.html.

[3] B. Amento, B. Balasubramanian, R. J. Hall, K. Joshi, G. Jung, and K. H. Purdy. 2016. FocusStack: Orchestrating Edge Clouds Using Location-Based Focus of Attention. In *ACM Symposium on Edge Computing (SEC'16)*.

[4] Shabab Bazrafkan and Peter M Corcoran. 2018. Pushing the AI envelope: merging deep networks to accelerate edge artificial intelligence in consumer electronics devices and systems. *IEEE Consumer Electronics Magazine* 7, 2 (2018), 55–61.

[5] Rudolf Beran et al. 1977. Minimum Hellinger distance estimates for parametric models. *The annals of Statistics* 5, 3 (1977), 445–463.

[6] Ketan Bhardwaj, Ming-Wei Shih, Pragya Agarwal, Ada Gavrilovska, Taesoo Kim, and Karsten Schwan. 2016. Fast, Scalable and Secure Onloading of Edge Functions using AirBox. In *Proceedings of the 1st IEEE/ACM Symposium on Edge Computing (SEC'16)*.

[7] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 443–448.

[8] Léon Bottou. 1998. Online Algorithms and Stochastic Approximations. (1998). http://leon.bottou.org/papers/bottou-98x revised, oct 2012.

[9] Leo Breiman. 1996. Bagging predictors. *Machine Learning* 24, 2 (01 Aug 1996), 123–140. https://doi.org/10.1007/BF00058655

[10] Ignacio Cano, Markus Weimer, Dhruv Mahajan, Carlo Curino, Giovanni Matteo Fumarola, and Arvind Krishnamurthy. 2017. Towards Geo-Distributed Machine Learning. *IEEE Data Eng. Bull.* 40, 4 (2017), 41–59. http://sites.computer.org/debull/A17dec/p41.pdf

[11] Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. 2008. Coordinate Descent Method for Large-scale L2-loss Linear Support Vector Machines. *Journal of Machine Learning Research* 9 (2008), 1369–1398. https://dl.acm.org/citation.cfm?id=1442778

[12] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System.. In *OSDI*, Vol. 14. 571–582.

[13] Radha Chitta, Rong Jin, and Anil K Jain. 2012. Efficient kernel clustering using random fourier features. In *2012 IEEE 12th International Conference on Data Mining*. IEEE, 161–170.

[14] Yi-Min Chou, Yi-Ming Chan, Jia-Hong Lee, Chih-Yi Chiu, and Chu-Song Chen. 2018. Unifying and Merging Well-trained Deep Neural Networks for Inference Stage. *CoRR* abs/1805.04980 (2018). arXiv:1805.04980 http://arxiv.org/abs/1805.04980

[15] Cisco. 2017. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper. https://cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf

[16] Stephane Daeuble. [n.d.]. Small cells and Mobile Edge Computing cover all the bases for Taiwan baseball fans. https://www.nokia.com/blog/small-cells-mobile-edge-computing-cover-bases-taiwan-baseball-fans/.

[17] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2007. Boosting for Transfer Learning. (2007), 193–200. https://doi.org/10.1145/1273496.1273521

[18] Facebook. [n.d.]. Applying machine learning science to Facebook products. https://research.fb.com/category/machine-learning/.

[19] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Comput. Surv.* 46, 4 (2014), 44:1–44:37. https://doi.org/10.1145/2523813

[20] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. 2008. A dual coordinate descent method for large-scale linear SVM. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008 (ACM International Conference Proceeding Series)*, William W. Cohen, Andrew McCallum, and Sam T. Roweis (Eds.), Vol. 307. ACM, 408–415. https://doi.org/10.1145/1390156.1390208

[21] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds.. In *NSDI*. 629–647.

[22] Ke-Jou Carol Hsu, Ketan Bhardwaj, and Ada Gavrilovska. 2019. Couper: DNN Model Slicing for Visual Analytics Containers at the Edge. In *4th ACM/IEEE Symposium on Edge Computing (SEC'19)*.

[23] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile edge computing—A key technology towards 5G. *ETSI white paper* 11, 11 (2015), 1–16.

[24] Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 97–106.

[25] Tetsuya Ishikawa. 2019. Random Fourier Features. https://github.com/tiskw/RandomFourierFeatures

[26] Hyuk-Jin Jeong, Hyeon-Jae Lee, Chang Hyun Shin, and Soo-Mook Moon. 2018. IONN: Incremental Offloading of Neural Network Computations from Mobile Devices to Edge Servers. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 401–411.

[27] Ruoming Jin and Gagan Agrawal. 2003. Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 571–576.

[28] James M Joyce. 2011. Kullback-leibler divergence. In *International encyclopedia of statistical science*. Springer, 720–722.

[29] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices* 52, 4 (2017), 615–629.

[30] Sebastian Kauschke and Johannes Fürnkranz. 2018. Batchwise Patching of Classifiers. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[31] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. 2004. Detecting change in data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 180–191.

[32] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. 2017. SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization. In *International Conference on Machine Learning*. 1866–1874.

[33] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[34] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. 2014. Mondrian forests: Efficient online random forests. (2014), 3140–3148.

[35] Yann LeCun. 2010. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (2010).

[36] Dawei Li, Theodoros Salonidis, Nirmit V Desai, and Mooi Choo Chuah. 2016. Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition. (2016), 64–76.

[37] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI*, Vol. 14. 583–598.

[38] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. 14 (2014), 583–598.

[39] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI*, Vol. 14. 583–598.

[40] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Trans. Information Theory* 37, 1 (1991), 145–151. https://doi.org/10.1109/18.61115

[41] Dirk Lindemeier. 2015. Nokia: EE and Mobile Edge Computing ready to rock Wembley stadium. https://www.nokia.com/blog/ee-mobile-edge-computing-ready-rock-wembley-stadium/.

[42] Viktor Losing, Barbara Hammer, and Heiko Wersing. 2018. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing* 275 (2018), 1261–1274. https://doi.org/10.1016/j.neucom.2017.06.084

[43] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. 2012. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment* (2012), 716–727.

[44] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* 46, 253 (1951), 68–78.

[45] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaíz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recognition* 45, 1 (2012), 521–530. https://doi.org/10.1016/j.patcog.2011.06.019

[46] IoT Now and Sheetal Kumbhar. 2017. Intel, RIFT.io, Vasona Networks and Xaptum to Demo IoT Multi-Access Edge Computing. https://tinyurl.com/intel-riftio-Vasona-Xaptum.

[47] Opinov8. 2019. How Do Amazon, Facebook, Apple and Google Use AI? https://opinov8.com/how-do-amazon-facebook-apple-and-google-use-ai/.

[48] Anand Padmanabha Iyer, Li Erran Li, Mosharaf Chowdhury, and Ion Stoica. 2018. Mitigating the Latency-Accuracy Trade-off in Mobile Data Analytics Systems. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 513–528.

[49] Sinno Jialin Pan, Qiang Yang, et al. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[50] Manohar Parakh. 2018. How Companies Use Machine Learning. https://dzone.com/articles/how-companies-use-machine-learning.

[51] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830. http://dl.acm.org/citation.cfm?id=2078195

[52] Michele Polese, Rittwik Jana, Velin Kounev, Ke Zhang, Supratim Deb, and Michele Zorzi. 2018. Machine Learning at the Edge: A Data-Driven Architecture with Applications to 5G Cellular Networks. *CoRR* abs/1808.07647 (2018). arXiv:1808.07647 http://arxiv.org/abs/1808.07647

[53] Ali Rahimi and Benjamin Recht. 2008. Random features for large-scale kernel machines. In *Advances in neural information processing systems*. 1177–1184.

[54] Jon NK Rao and Alastair J Scott. 1981. The analysis of categorical data from complex sample surveys: chi-squared tests for goodness of fit and independence in two-way tables. *Journal of the American statistical association* 76, 374 (1981), 221–230.

[55] Pablo Rodriguez. 2017. The Edge: Evolution or Revolution. In *ACM/IEEE Symposium on Edge Computing (SEC'17)*.

[56] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. 2009. On-line random forests. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE, 1393–1400.

[57] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[58] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. 2014. Cloudlets: at the leading edge of mobile-cloud convergence. In *2014 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*. IEEE, 1–9.

[59] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.. In *ICISSP*. 108–116.

[60] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[61] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. 2017. Federated multi-task learning. In *Advances in Neural Information Processing Systems*. 4424–4434.

[62] Heng Wang and Zubin Abraham. 2015. Concept drift detection for streaming data. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*. 1–9. https://doi.org/10.1109/IJCNN.2015.7280398

[63] Shuo Wang, Leandro L. Minku, Davide Ghezzi, Daniele Caltabiano, Peter Tiňo, and Xin Yao. 2013. Concept drift detection for online class imbalance learning. In *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*. 1–10. https://doi.org/10.1109/IJCNN.2013.6706768

[64] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2018. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. (2018), 63–71.

[65] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.

[66] Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri. 2017. Towards efficient edge cloud augmentation for virtual reality MMOGs. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 8.

[67] Liming Zhao, Jingdong Wang, Xi Li, Zhuowen Tu, and Wenjun Zeng. 2016. Deep convolutional neural networks with merge-and-run mappings. *arXiv preprint arXiv:1611.07718* (2016).