DCP: a TCP-Inspired Method for Online Domain Adaptation under Dynamic Data Drift

Alessandro Buratto*, Marco Levorato†, and Leonardo Badia*
∗ Dept. of Information Engineering (DEI), University of Padova, Italy
email: {alessandro.buratto.1@phd. , leonardo.badia@}unipd.it
† Donald Bren School of Information and Computer Science, Irvine, California
email: levorato@uci.edu

***Abstract*—Mobile computing faces challenges due to the resource constraints of mobile devices, such as limited computing power, energy, and connectivity. These limitations hinder the use of high-complexity classifiers and wireless transmissions. To address this issue, we propose a novel collaboration paradigm between mobile devices and edge servers, where the edge server assists the mobile devices by dynamically retraining a low-complexity classifier to adapt to temporal changes in data distribution. We propose a novel approach called drift control protocol (DCP) which is inspired by TCP congestion control mechanism. DCP aims to strike a balance between low-complexity classifier retraining frequency and communication costs with the edge server. It adjusts the update rate of the classifier on the mobile device based on distribution drift characteristics and controls the number of input samples sent to the edge server to improve accuracy. We evaluate and study different versions of DCP using synthetic and real datasets We demonstrate that DCP keeps the error bound, while reducing the burden of the communication cost by 90% for the mobile nodes, which makes our proposal suitable for online domain adaptation.**

***Index Terms*—Online Domain Adaptation, Data Drift, TCP, Edge computing**

I. Introduction

Machine learning (ML) is becoming an increasingly central component of a broad spectrum of mobile applications [1], ranging from autonomous vehicles and mobile health to smart manufacturing [2]–[4]. However, as the complexity of the input data and algorithms grows, the demands of such applications clash with the limited capabilities and resources of mobile devices, e.g., computing power and energy reservoir. This can be mitigated through edge computing [5]–[7], where an infrastructure level compute-capable devices – the edge server (ES) – takes over the execution of the ML algorithms. However, this necessitates the transfer of information-rich data streams over capacity constrained wireless links that are likely shared by multiple mobile devices (MDs) [8], [9].

To address this impasse, a small number of recent contributions propose a different paradigm for the MD-ES collaboration, where the ES actively assists the MDs in dynamically *adapting* low-complexity classifiers that are specifically trained to focus on the current – and hopefully more localized – distribution of input data perceived [10]–[13]- which is often referred to as *context*. This offers the benefit that the MD can use a feasible ML model to process the input data flow, or a portion of it, thus decreasing the volume of data sent to the ES, and possibly reducing latency.

The key to the success of this strategy is the ability of the low-complexity classifier (which we refer to as local classifier in the following) to adapt to the specific operational context of the MD, which is likely associated with a more focused distribution of the input data [14], [15]. Thus, such approaches refer to the general area of domain adaptation [16], but transpose this concept in the realm of online, rather than offline, operations of the system. The hope is that by adapting the parameters of the local classifier to localized input distributions that are drifting over time the MD can achieve high task performance in spite of resource and channel constraints.

However, such an architecture may still suffer from problems when the domain changes over time [17], [18], which we try to address in the present paper. The proposed architecture would be efficient as long as the local domain at the MD is precisely defined. The presence of a dynamic domain drift may alter the local classification and make it inaccurate. One of the core technical challenges, then, is to detect and precisely quantify the drift. This calls for also monitoring the drift at the ES, which must be done carefully since it may negate the principle of the architecture to avoid a frequent supervised intervention of the ES.

For this reason, in this paper we propose a domain control protocol (DCP), which, as the name itself suggests, is inspired by the well-known transmission control protocol (TCP), used in the transport layer of the Internet suite to dynamically adjust the amount of exchanged data across an end-to-end connection [19]. Our purpose is to detect the need for a retraining of the local classifier at the MD, performed by the ES, and force an update of its parameters, by relying on minimal information and with low frequency. Compared with a periodic update of the local classifier, we show that our approach is able to detect increasingly drifting data, in a similar way to the congestion detection characteristics of TCP, and reduce the exchanges by limiting them to when they are needed. As a result, we obtain

This work was supported by the Italian PRIN project 2022PNRR "DIGIT4CIRCLE" and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART")

an accurate classification even in the presence of a variable domain drift, without resorting to constantly retrain the local classifier.

In more detail, our contribution is two-fold. First, despite the true extent of the domain drift being inaccessible to the local classifier, we mathematically derive an upper bound that allows for a mechanism akin to congestion control, with additive increase multiplicative decrease properties, that intensifies the training of the local classifier following strong domain drifts. Second, we formalize the DCP procedure, for which we can actually identify variants, aptly named Tahoe and Reno since they follow the same rationale of their counterparts in TCP. We perform comparative evaluations of DCP techniques with other alternative benchmarks, namely constant-interval updates, Q-learning based approach and the Drift Adaptive Deep Reinforcement Learning based Scheduling (DA-DRLS) technique [20], in both synthetic datasets that simulate various types of drifts with tunable parameters and the room occupancy dataset described in [21]. We prove that DCP is a practical low cost solution for real implementations, as it achieves a better cost vs. accuracy trade off with a 90% cost reduction with respect to a 1-persistent update schedule.

The rest of this paper is organized as follows. In Section II, we review related work. Section III presents the model of the computing architecture with edge offloading and domain adaptation, introducing the problem of dynamic domain drift; inspired by formal theoretical results, we introduce DCP as a practical approach to address it. Section IV contains the comparative performance evaluation of DCP and other different approaches. Finally, Section V concludes the paper.

II. Related Work

Domain adaptation is an umbrella term for different meta-optimization techniques used to adjust global knowledge to a specific target domain with special distribution characteristics [22], since the data used to train a model may be different from what is encountered at run time.

Such techniques are relevant in the context of multimedia applications, where domain adaptation is necessary to address the differences in data distribution due to different ambient conditions of images, videos, and audio content [14]. For example, a machine learning model trained on images captured in a well-lit environment may not perform as well when applied to images captured in low-light conditions. Domain adaptation can adapt the model to the new distribution of data encountered in the target domain [23]. Domain adaption is also relevant for natural language processing [24], as contextualized word embedding techniques may provide a more expressive interpretation of words, which is very relevant for sentiment analysis or entity recognition. Finally, a field where the fine-tuning of a classifier on a local domain received considerable recent interest is that of chemical sensors [3], [25], since monitoring of analytes and recalibration to different environments is important in smart healthcare, manufacturing, and pharmaceutical industry.

In IoT scenarios, domain adaptation can similarly play an important role in optimizing the performance of ML models deployed on devices with low computational and storage capabilities, where it is often not feasible to train and deploy complex models. Thus, models can be trained on more powerful servers and then adapted to the target domain. This would account for the specific characteristics of the data encountered by the MD, such as sensor measurements from IoT devices, so as to obtain high accuracy and reliability even on devices with limited capabilities.

When adapting the domain to what observed in a specific context, the local data distribution is commonly referred to as target domain or drift distribution [25]. The literature is abundant with proposals for adjusting a source domain to its drifted version, essentially consisting of a layered application of ML with multiple low-complexity techniques being regarded as available at the local classifier, as well as performing domain adaptation [26], [27].

Reference [23] proposes a unified framework to analyze domain adaptation techniques from a holistic perspective. Its focus is on multimedia applications, where the problems mostly consist of the unavailability of annotated data, which are impractical to obtain for many different domains [15], as well as perform domain adaptation when the drift distribution is unknown, which calls for state of the art techniques for unsupervised learning [22]. All of these issues pertain to data availability, not to their dynamic change over time or the bottlenecks created in the network infrastructure.

In general, domain adaptation is mostly performed in the literature to simplify learning in the presence of abundant data with different contexts [17], [24], [26], [28]. The dynamic acquisition of these data is not present, or anyways not a concern as there are no real-time features to account for the time dynamics, just a loose requirement for low complexity. There are a few notable exceptions in this sense [18], [25], where the domain drift is dynamic and time constraints are present to adjust the domain in the fastest possible way. This is especially the case for industrial applications, e.g., detection of chemical hazards [3].

Chowdhury et al. [20] propose DA-DRLS which is a deep reinforcement learning solution to a drift in demand through time in a dense IoT scenario. However, in this case the objective clashes with making a sparing use of the computational resources, as is instead our concern, motivated by the edge offloading architecture [5], [6]. In [10], the authors use a small "pioneer" neural network to select the neural network used for the machine learning task. In other words, the few existing works accounting for with the time dimension of the drift compensation are trying to catch up with it as rapidly as possible, whereas we argue that a pervasive computing architecture would need instead to track it precisely, yet within the limits of not overloading the channel [12].

This explains why we look for solutions that, in agreement with the rationale of computational intelligence in IoT [2], perform limited exchanges of information but try exploiting them in the best possible way.

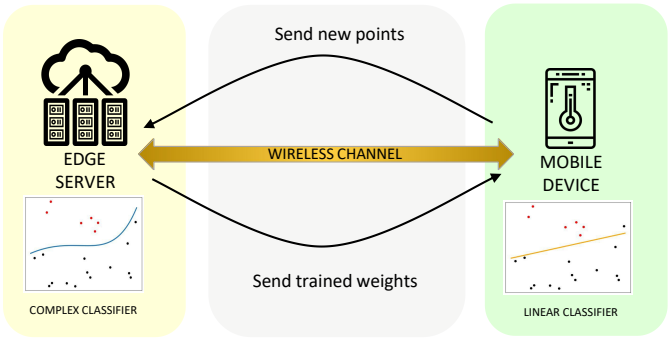

Fig. 1. System model. An ES and an MD are connected via a wireless channel. The ES sends model weights to the MD and the latter sends back new measured points.

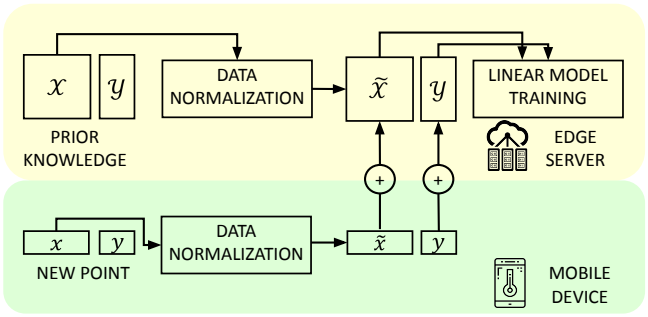

Fig. 2. Operation schematic. The ES uses previously evaluated points as a prior and incrementally adds new points, whenever the MD sends them. This incremental database is used to train the best possible linear classifier.

III. Model and Proposed Approach

We consider a mobile computing scenario with two main actors: a resource-constrained MD sensing the environment and analyzing the collected data, and a compute-capable ES connected through a wireless link [6] (see Fig. 1). The MD is tasked with the classification of data points acquired by its internal sensors, that acquire information from heterogeneous data sources with different distribution across them. Due to its resource limitations, the MD leverages edge offloading to the more powerful ES.

Formally, we consider m to be the number of sensors the MD is equipped with and n is the total number of points the MD has access to at the end of the procedure. We denote with $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x}_i\}_{i=1}^{n} \subseteq \mathbb{R}^{n\times m}$ the set of the points measured by the sensor, where $\boldsymbol{x}_i \in \mathbb{R}^m$ are the single measurements, and $\boldsymbol{\mathcal{Y}} = \{y_i\}_{i=1}^{n} \subseteq \{0,1\}^{n\times 1}$ as the set of binary labels associated with each point. Fig. 2, illustrates the normalization of the parameter space, after which we obtain $\tilde{\boldsymbol{\mathcal{X}}} = \{\tilde{\boldsymbol{x}}_i\}_{i=1}^{n} \subseteq [0,1]^{n\times m}$ where all the features are normalized within $[0,1]$.

Due to its limited capabilities, the MD can support the execution of low-complexity classifiers, which here we set to be a linear model ℓ, e.g., a linear support vector machine (SVM) [7]. In many settings and tasks, such low-complexity classifier results in low performance when applied to general input distributions. Conversely, the ES is implementing a high-complexity classifier f capable of achieving high classification accuracy. More formally, let $\mathcal{F}$ be the set of all the possible classifiers and $\mathcal{A}$ a function that evaluates the accuracy of a model, f can be defined as

$$f = \text{argmax}_{c\in\mathcal{F}}\mathcal{A}(c). \tag{1}$$

Because its accuracy is optimal, we can consider it to be our ground truth. Similarly ℓ is the best linear approximation of f. Formally, let $\mathcal{L}$ be the set of all the linear classifiers approximating f

$$\ell = \text{argmax}_{l\in\mathcal{L}}\mathcal{A}(l). \tag{2}$$

In the collaboration paradigm we propose, the ES performs online parameter adaptation of the linear model of the MD to enable an efficient local classification. Thus, the ES optimizes the parameters $\boldsymbol{m}$ and $\boldsymbol{q}$, that is, the vector of the slope and intercept of the m-dimensional linear decision function, respectively. Because ℓ is derived from f and the latter is arbitrarily complex, the local classifier introduces an error. To mitigate this issue, in addition to parameter adaptation, the MD can offload some data points to the ES. The choice of the points to offload is left to the MD itself which selects them in areas where its classification is less accurate (e.g., in the region between the decision boundary and the support vectors in an SVM). Both updating the classifier and offloading data operations have a cost (e.g., bandwidth usage), and should be kept to a minimum.

We consider slotted time, where the slots have the same duration equal to the round trip time (RTT) of the channel between MD and ES, independent of whether the MD starts a communication attempt or not. All the communications between the MD and the ES are always initialized by the MD at the beginning of the slot and end with the reply of the ES, which marks the end of the time slot. For the remainder of the paper, we will use "round" instead of "slot" when it is important to include the decision of the MD to start a communication.

A. Dynamic Data Drift

Standard approaches to domain adaptation usually consider a different context for the local classifier that needs to be adapted only once [22]–[24]. Conversely, we assume that the MD is placed within a dynamic environment, and for this reason its data are subject to changes in distribution over time. This phenomenon will hereby be referred to as the *online domain drift*. Because of the continuous changes in the data distribution, there is the need for the MD to periodically update its internal classifier. In order to do so, the MD collects measured samples and classifies them, at the beginning of each round the MD decides whether to offload to the ES samples that are very close to the decision boundary $\tilde{x}$ and asks for the linear model to be retrained. These points are then used by the ES to update its own internal classifier f and to produce a

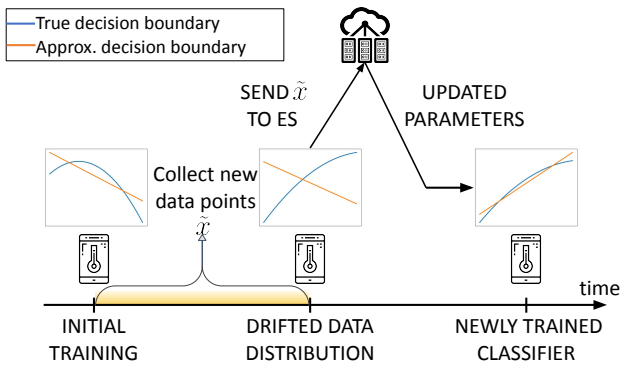

Fig. 3. Time evolution of the local classifier model. Upon request by the MD, the ES can retrain it and send the new parameters to the MD.

newer version of the linear classifier ℓ to be sent to the MD, see Fig. 3 for a visual representation. We use ℓ_i and f_i to indicate the linear model and the complex classifier at round i. We define the error introduced by the linear approximation as

$$E_{i,k} = \int \|f_i(\tilde{\boldsymbol{x}}) - \ell_k(\tilde{\boldsymbol{x}})\| d\tilde{\boldsymbol{x}} \tag{3}$$

and we also define the error made by the MD as the difference of its most recent classifier and the previous one

$$e_i = \int \|\ell_i(\tilde{\boldsymbol{x}}) - \ell_k(\tilde{\boldsymbol{x}})\| d\tilde{\boldsymbol{x}}, \text{with } k < i. \tag{4}$$

Because f changes through time it holds that $E_{i,k} > E_{i,i}$. We aim to jointly minimize two utility functions: the error made by the linear model and the total cost of updates. Because the first one is steadily increasing and concave and the second one is also steadily increasing and convex, we need to design update policies that find Pareto efficient solutions.

Note that the MD does not know the accuracy of its local classifier it can only infer the current error by using the most recent value of e_i.

B. Main findings

The purpose of this subsection is to prove some theoretical motivations for the proposed procedure, i.e., that the MD can quantify the magnitude of the drift by comparing the linear model parameters between different rounds.

We preliminary observe that the MD ought to pre-process the raw data coming from the sensors by normalizing its input multi-dimensional data, to have coherent representation in all dimensions.

We further state the following, in order to have a non-trivial classification problem.

Definition 1. At any time instant at least one point for each category is present in the considered space.

Otherwise, the classifier would have a perfect accuracy as there would be only a category to choose from.

Now, let $\mathcal{M} \subseteq [0,1]^m$ be a m-dimensional Borel space [29]. Let $f_i : \mathcal{M} \to \mathbb{R}$ be an arbitrary decision function and $\ell_i : \mathcal{M} \to \mathbb{R}$ be the corresponding best linear approximation at round i. Let $\int_{\mathcal{M}} \|\cdot\|$ be the measure applied to measurable functions f_i and ℓ_i. We can exploit the following results.

Theorem 1. *The difference (i.e., the error) between f_i and the linear approximation ℓ_i is lower bounded by the difference between f_i and the linear approximation obtained in the previous round ℓ_{i-1}, and upper bounded by the same quantity plus the difference between the linear approximations over two subsequent slots, i.e., ℓ_{i-1} and ℓ_i. The latter is an upper bound of the increment in the difference over consecutive slots.*

Proof: For indices i and $k < i$, consider

$$L_{i,k} = \int_{\mathcal{M}} \|\ell_i(\tilde{\boldsymbol{x}}) - \ell_k(\tilde{\boldsymbol{x}})\| d\tilde{\boldsymbol{x}} \tag{5}$$

The first part of the theorem using the notation of (3) states that

$$E_{i,i} \leq E_{i,i-1} \leq E_{i,i} + L_{i,i-1}\,, \tag{6}$$

which is immediate since the first inequality is a consequence of ℓ_i being the best linear approximation of f_i, thus $E_{i,i} = \min_k E_{i,k}$, whereas the second part holds due to the triangular inequality. Also, through an algebraic manipulation of subtracting $E_{i,i}$, we get

$$0 \leq E_{i,i-1} - E_{i,i} \leq L_{i,i-1}\,. \qquad \blacksquare \tag{7}$$

Corollary 2. *Theorem 1 can be promptly extended to the case of non-consecutive slots.*

Proof: With the same definitions of (3) and (5), and iterating the reasoning, one can promptly get

$$0 \leq E_{i,k} - E_{i,i} \leq L_{i,k}\,. \qquad \blacksquare \tag{8}$$

In the previous result, the ground truth f_i is not known to the MD, but linear approximations possibly are, as they are received from the ES at updating instants. The MD cannot precisely evaluate the goodness of the current linear approximation, it just knows it to be the best possible choice made by the ES at the time of the update, nor it can assess the error due to using an old linear estimate ℓ_k. However, it can exploit $L_{i,k}$ to bound the increase in the error due to using the old estimate, which is an indicator of how fast the domain is drifting. This is what prompts us to a general approach for domain adaptation strategies, where $L_{i,k}$ gauges the frequency of requests for retraining the linear classifiers, depending on this drift. These theoretical results imply that the drift in the data will be bounded.

Theorem 3. *Given an m-dimensional Borel space $\mathcal{M} \subseteq [0,1]^m$ and a set of n points sampled from $\mathcal{M}$ $\tilde{\boldsymbol{\mathcal{X}}} = \{\tilde{\boldsymbol{x}}_i\}_{i=1}^n \subseteq [0,1]^{n \times m}$ with a set of binary labels $\boldsymbol{\mathcal{Y}} = \{y_i\}_{i=1}^n \subseteq \{0,1\}^{n \times 1}$ changing through time, the hyperarea (i.e. the error) included between two hypersurfaces that perfectly separate the two categories at any given time instant is finite.*

Proof: Assume that there exists a pair of linear hyperplanes whose distance measure is infinite, i.e., $L_{i,k} = \infty$. This can only happen when the separating hyperplanes coincide

with the opposite parallel bounds of the Borel space $\mathcal{M}$. Thus, all points inside the considered space are in the same category. This contradicts Definition 1, thus the separating hyperplanes have finite slope and $L_{i,k} < \infty$. By Corollary 2 also the difference of the hypersurfaces is bounded. ■

This implies that a periodic updating of the classifier achieves bounded average error in the presence of a drift, as a direct consequence of Theorem 3. Yet, we can do better than a periodic updating, in the sense of achieving the same error with fewer updates, by tracking the drift through the estimate given by the difference of the linear approximations. In the following theorem, we prove that tracking the linear approximations keeps the drift within the same bounds of a periodic sampling (but with fewer updates).

Theorem 4. *An aperiodic sampling triggered by the violation of a bound on the difference between the linear approximations achieves bounded drift. The periodic sampling achieving the same average drift performs on average more updates.*

Proof: Note that we compare a periodic and an aperiodic update of the classifier. We follow (5) to denote their error terms, but we add superscripts (p) and (a), respectively, to distinguish between them.

Consider a periodic updating with period P and error bounded to ε, i.e., $E^{(p)}_{i,k} < \varepsilon$ for $k-i < P$. Take $\mathbb{E}\left[E^{(p)}_{i,i}\right] = \varphi < \varepsilon$. Define an aperiodic updating that triggers a retraining from the ES whenever the error of the linear approximation $L^{(a)}_{i,k} \geq \varepsilon - \varphi$ and $k-i \geq P$. By construction, the aperiodic updating retrains the classifier less frequently than the periodic one, and according to Corollary 2, $\mathbb{E}\left[E^{(a)}_{i,k}\right] < \varepsilon$. ■

All of these results prove that tracking the difference of the linear approximations is a good indicator of how the domain is drifting, and less expensive than a periodic retraining. The problem is that we do not have the linear approximation available at every round, unless we perform a retraining with period 1, which would invalidate the whole procedure. Thus, we devise practical strategies to make good use of the linear approximations when available (i.e., when the retraining is actually performed), which leads to the development of DCP.

C. DCP Algorithms

We take inspiration from the congestion control mechanism of the TCP protocol [19] in designing an online policy to follow the domain drift. The main rationale behind this transport layer protocol is that it is possible to send multiple packets across a communication link without waiting for the relative acknowledgement packet to be received by the sender before a new packet is sent over the channel. This time interval is commonly referred to as the round trip time (RTT).

The goal of TCP congestion control procedure is to automatically adjust the number of segments sent in an RTT to the network state. To this end, TCP uses two key parameters: the congestion window (*cwnd*) that keeps track of the MSSs sent over the communication channel at each RTT and the slow start threshold (*ssthresh*) which indicates the end of the slow start phase. Slow start is the first phase of the protocol where the *cwnd* grows exponentially in the number of passed RTTs. When the congestion window reaches the slow start threshold, the former keeps growing linearly at each RTT according to the number of received acknowledgements.

The setup of these parameters depends on different error situations that can occur and are interpreted as problems due to congestion. For example, a packet can be lost and the receiver never receives an acknowledgement (timeout error); or, the receiver keeps receiving the same acknowledgment for 3 times (3-DUPACK). The main difference between versions of TCP lies in how the protocol handles these events.

Our rationale is similar, in that we need to adjust the number of rounds between two requests for retraining the linear classifier. Ideally, if we detect that the domain drift is static, i.e. the change is constant over time or it is completely nonexistent, we can perform a sparser retraining. Conversely, if a fast dynamic drift is perceived, the frequency of retraining must be increased, to avoid that the linear classifier is not accurate at all. Of course, this is adjusted to a different scenario, since we consider rounds of the classifier as opposed to segments, and especially we need to define the condition considered as erroneous that triggers the congestion control procedures.

In light of the previous results, upon performing an update at round i, we consider the error between the current linear approximation and the previous one performed at round k, i.e., $L_{i,k}$. Moreover, we introduce a system parameter called *precision threshold*, just referred to as "threshold" in the remainder of the paper and denoted as θ, that indicates the maximum discrepancy between the linear decision models over different updates that we are allowing before forcing a new one. In other words, the threshold is violated whenever $L_{i,k}$ exceeds θ.

We propose three procedures to update the linear model of the mobile device:

DCP Tahoe: The MD requests an update to the ES according to the evolution of the congestion window (*cwnd*) in the TCP Tahoe protocol. If the threshold is violated, we treat this as a TCP loss event, which results in halving the slow start threshold and resetting the *cwnd* to 1 round. In practice, we restart from the beginning every time an error occurs, while reducing the duration of the slow start phase of the protocol until we are in the linear phase after each restart.

DCP Reno: The approach is similar to DCP Tahoe, but we consider the case of an error as receiving three duplicate acknowledgements in TCP Reno, meaning the slow start threshold is halved and the congestion window is set to *ssthresh* plus 3 units. Thanks to this, the number of update requests after an error event is reduced, as the updating policy always waits at least 4 rounds before attempting a new update.

Moving average DCP: This is meant to automatically set the threshold θ. At the start of the procedure, and at randomly chosen instants thereafter, the method performs a series of consecutive updates to estimate the mean error made by the approximation, then the result is fed to a DCP Tahoe.

IV. Performance Evaluation

A. Key Performance Indicators

We ought to evaluate the performance of a procedure aimed at finding the best update schedule with the following key performance indicators (KPI):

Mean error: the mean of the errors made across all rounds. It serves to grasp the ability of the techniques to follow the drift during the whole experiment. Let e_i be the error performed at round i and let N be the total number of rounds in the considered experiment:

$$\bar{e} = \frac{1}{N} \sum_{i=1}^{N} e_i \tag{9}$$

Cumulative mean error: the cumulative mean of the error made during the experiment, averaged over multiple experiments, to limit the effect of random parameters generation at model initialization and, in the specific case of the synthetic dataset, to reduce the variability of different sequences of weights generated with the same parameters. Let K be the total number of experiments and $e_i^{(k)}$ be the error performed in the i-th round in the k-th experiment. For round n the cumulative mean error takes the form:

$$\bar{\mathcal{E}}_n = \frac{1}{Kn} \sum_{k=1}^{K} \sum_{i=1}^{n} e_i^{(k)} \tag{10}$$

Cumulative standard deviation error: the cumulative standard deviation of the cumulative mean error made across the whole duration of the experiment. It is also the result of a further mean across different experiments to limit the effect of random generation. Let $\bar{e}_{1:n}^{(k)}$ be the mean error from round 1 to round n in k-th experiment:

$$\bar{\sigma}_n = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n} \sqrt{\frac{\left(e_i^{(k)} - \bar{e}_{1:n}^{(k)}\right)^2}{n}} \tag{11}$$

Cumulative cost: the cumulative sum of all the instances when the MD asks for a model update at the ES. We assume the same unitary cost per each update indicated by the indicator function $\mathbb{1}(\cdot)$.

$$\mathcal{C} = \sum_{i=1}^{N} \mathbb{1}(\ell_i \neq \ell_{i-1}) \tag{12}$$

Incremental cost: the first order discrete derivative of the cumulative cost. It shows the expected cost of a round at different times during the execution of the policy. To avoid noisy plots, and since it serves as an average indicator, we actually consider its mean value over a sliding window of size k. We further define $\mathcal{C}_{1:i}$ which is the cumulative cost from round 1 to round i.

$$\nabla \mathcal{C}_n = \frac{1}{k} \sum_{i=n-k+1}^{n} (\mathcal{C}_{1:i} - \mathcal{C}_{1:i-1}) \tag{13}$$

We now present and discuss the results obtained by applying the proposed methods in both synthetic and real datasets.

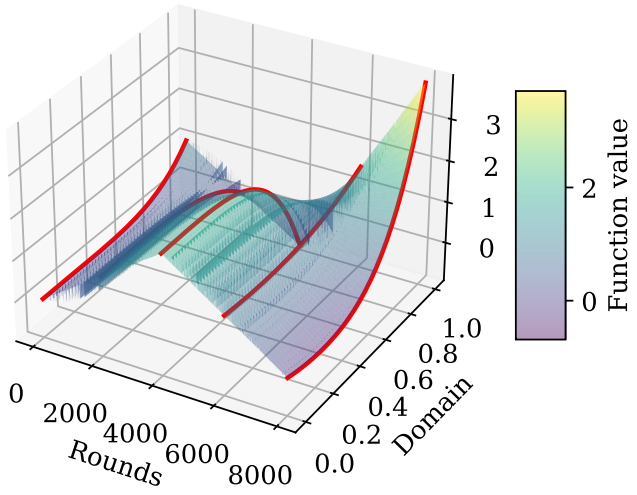

Fig. 4. Time evolution of a function generated for the synthetic dataset. The speed at which the function changes is not constant across rounds and there are some visible rapid changes, in particular in the first rounds. Bold red curves indicate the exact shape at a given round.

B. Datasets

We evaluate the proposed approaches for (a) a synthetic dataset, to test the ability of our methods to follow a controlled variation of the parameter space; (b) a real dataset, to test the performance in a concrete application scenario. The synthetic dataset is constructed by a linear combination of two randomly chosen multidimensional polynomial functions of a pre-defined degree. The weights assigned to each function at each time step are generated according to a parameterized arbitrary non continuous function with tunable parameters for the steepness and error rate by switching at any round to a different sequence of weights. One of such parameters is the correlation factor ρ that controls the probability that at any given time the weight is the same of the one at the previous round. This simulates, for instance, the position tracking of a user that is going from point A to B with some random errors introduced for example by a poor GPS connection. Another example may be a thermometer in a partly cloudy day that rapidly changes the measured temperature when exposed to direct sunlight. Fig. 4 shows the evolution through rounds of a synthetic 1D function. The change in magnitude through time is evident, meaning that we can simulate large drifts and in some regions the surface presents sudden jumps. This allows us to simulate unexpected changes in the data distribution due to erroneous readings of the sensors.

The real dataset [21] contains aggregated data from multiple sensors monitoring temperature, CO_2, humidity and light, to infer occupancy of a research lab. The data is presented as time series sampled every minute.

C. Benchmark Methods

We hereby describe the other methods we will compare our DCP procedures with.

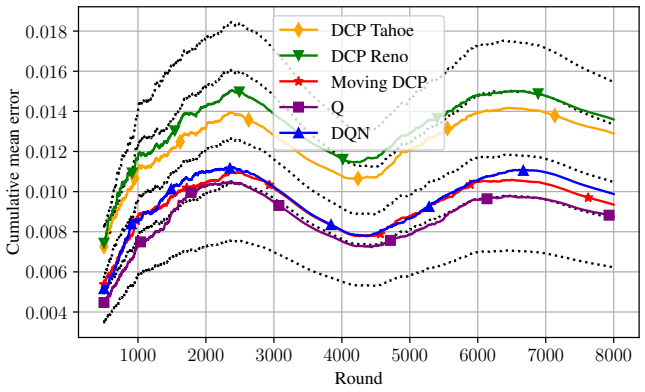

Fig. 5. Synthetic dataset: cumulative mean of the error. Dotted black lines represent the performance of constant interval updates from bottom to top every 1, 2, 3, 5 and 7 rounds.

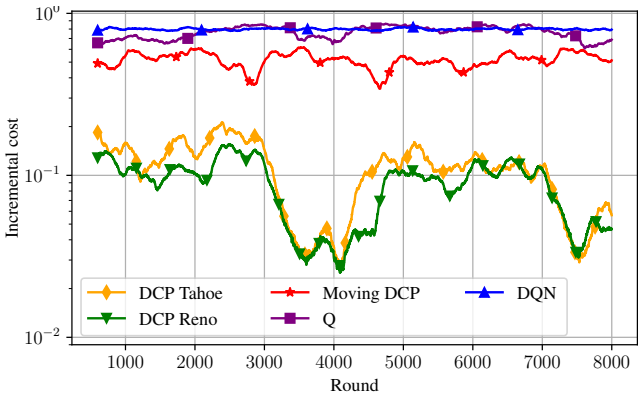

Fig. 6. Synthetic dataset: incremental cost averaged over a 200-round window.

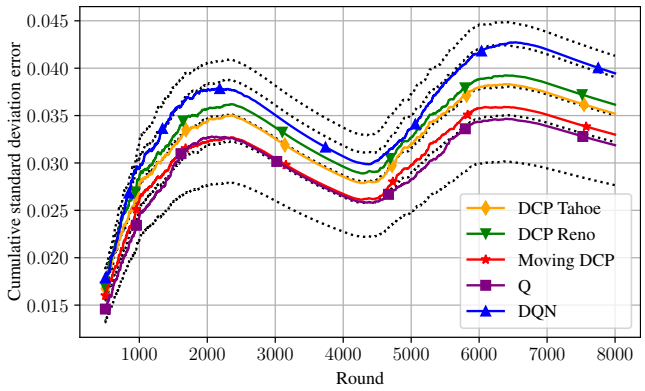

Fig. 7. Synthetic dataset: cumulative standard deviation of the error. Dotted black lines represent the performance of constant interval updates from bottom to top every 1, 2, 3, 5 and 7 rounds.

Q-learning: The choice of whether to update or not is left to a Q-learning agent which is trained with an epsilon-greedy policy over the course of the simulation. The learning process is described by a Bellman equation [30]

$$Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \quad (14)$$

where s_t and a_t are the state of the system and the action taken at time t respectively, $Q(s_t, a_t)$ is the Q-value given the state and the action pair, $\alpha \in [0, 1]$ is the learning rate that controls how much new information overrides the current one, r_t is the reward obtained by the agent at time-step t and $\gamma \in [0, 1]$ is the discount factor which is in charge of controlling the importance the model gives to future rewards. After some fine tuning, in our experiments we will use $\alpha = 0.3$ and $\gamma = 0.99$, which leads to the best minimization of the model error.

In the epsilon-greedy policy, we also need to set $\epsilon \in [0, 1]$ for the exploration-exploitation tradeoff of the agent, where $\epsilon = 0$ means that the agent always chooses the best action, and $\epsilon = 1$ forces the agent to choose an action at random, thus exploring the state space. We chose an exponentially decreasing ϵ to guarantee good exploration at the beginning of the procedure and more exploitation as the knowledge of the state space improves [31].

The agent is subject to a negative reward when it decides not to transmit which is proportional to the error performed by the model with respect to the last update and a less negative reward when it decides to perform an update. This choice is determined by the fact that we want to encourage the model to learn the optimal point at which the error made by the approximation is not too big without it being negatively impacted by the communication cost alone.

Deep Q-Learning: We implement the update strategy of DA-DRLS described in [20] adapting the reward to our model.

Constant-Interval Updates: We define constant update procedures that ask for updates of the linear model at constant interval. We use these methods as benchmarks to see how a genie-aided method would perform knowing the optimal constant update pattern to follow for the specific data drift.

D. Results

In this subsection, we present the graphs obtained by applying our proposed methods to the datasets introduced in Section IV-B. We compare our procedures with constant update policies, where the ES periodically retrains the MD after a fixed number of rounds. We perform this comparison with non-adaptive solutions, to highlight the ability of our protocols to autonomously adjust the frequency for updating the classifier. For our plots, we choose updates every 1, 2, 3, 5 and 7 rounds. These methods are reported in Figs. 8 and 12 as circle markers and with dotted lines in Figs. 5, 7, 9 and 11.

Figs. 5 and 7 show the cumulative mean error achieved by the different methods in the synthetic dataset. It is noticeable how all the methodologies are able to follow the changes in the error while automatically identifying the good range of constant updates to follow. Moving DCP, DQN and the Q-learning based method follow closely the constant update every two epochs, both from the mean and the standard deviation standpoint. This behaviour may suggest their advantage in following closely drifting domains while adapting accordingly. Conversely DCP Tahoe and Reno obtain values close to the constant update every 5 rounds, meaning that they follow the drift less closely than their competitors.

Fig. 6 reports the incremental cost for the different tech-

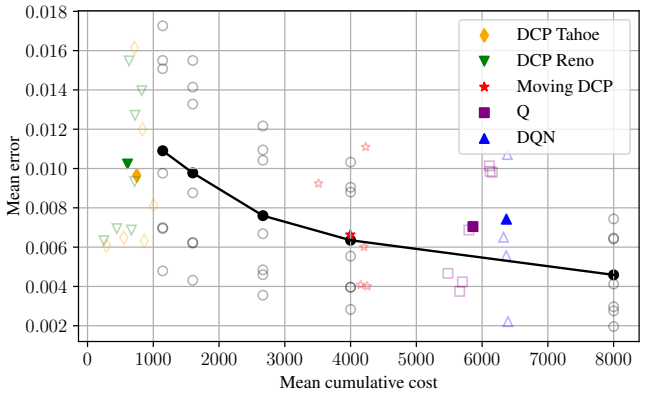

Fig. 8. Synthetic dataset: cost-error tradeoff.

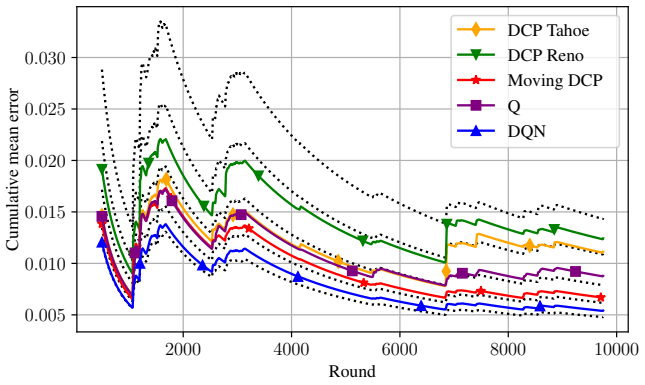

Fig. 9. Real dataset: cumulative mean of the error. Dotted black lines represent the performance of constant interval updates from bottom to top every 1, 2, 3, 5 and 7 rounds.

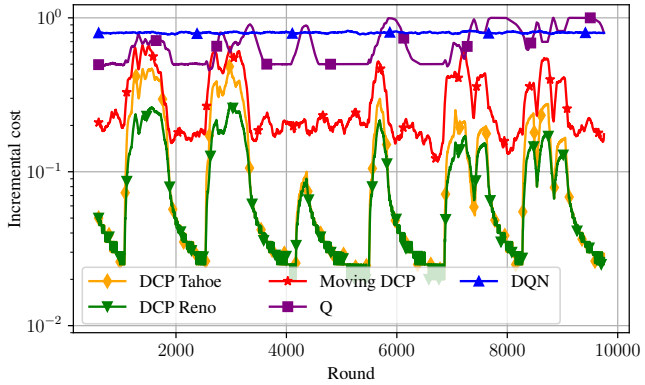

Fig. 10. Real dataset: incremental cost averaged over a 200-round window.

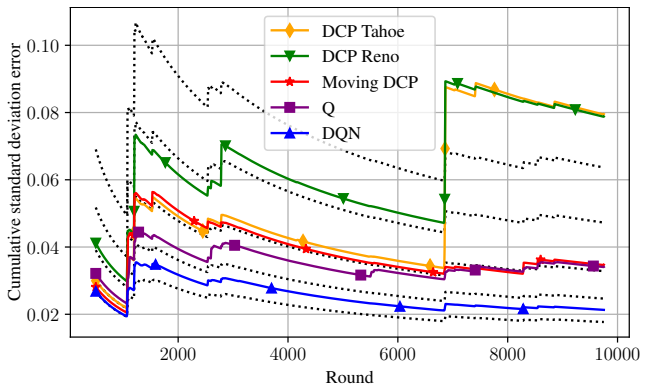

Fig. 11. Real dataset: cumulative standard variation of the error. Dotted black lines represent the performance of constant interval updates from bottom to top every 1, 2, 3, 5 and 7 rounds.

niques. It is evident that Q-learning, DQN and Moving DCP have a per round cost near 1 meaning that they require almost always an update. DCP Tahoe and Reno show a less constant cost change, meaning that they adapt the rate of update according to the drift changes during the experiment.

Fig. 8 shows the tradeoff between the mean error and the cumulative cost of the models. The Q-learning approach does not obtain low mean error values, like constant update techniques, even though its cumulative cost is comparable to making updates every 1 or 2 rounds. In this dataset also DQN shows similar performances, but with even higher cost neglecting the initial training. Moving DCP performs very similarly to a constant update every 2 rounds. DCP Tahoe and DCP Reno manage to keep the mean cumulative cost significantly lower (almost half) than the constant update methods and, as previously shown with Figs. 5 and 7, they bound the error to values comparable to constant update policies. This suggests that they can effectively keep a contained error even in the presence of a dynamic drift, while drastically reducing the global cost making them a viable solution to our task.

Figs. 9 and 11 show the mean cumulative error and standard deviation obtained by the different approaches in the real dataset. Also in this case all the methods effectively follow the mean error made by the drift. In this dataset, a big spike of the dynamic drift is in the near vicinity of round 7000. Thus, DCP Tahoe and Reno encounter a sudden increase in the error and the standard deviation, due to their attempt to decrease the update frequency as much as possible. However, it is important to notice that, despite this localized disruption, they recover very rapidly and converge to a stable trend.

Fig. 10 displays the incremental cost. It is evident that the DCP methods experience periods with fewer updates followed by bursty sections of fast updates. This behaviour is also typical of TCP when the network is strongly congested [19]. Q-learning and DQN show almost a constant update pattern meaning that they have issues in understanding when to effectively intensify the updates.

Fig. 12 shows the tradeoff between error and cost required for the policies. Q-learning approach proves to be too expensive and unable to bound the error when compared to constant updates. DQN achieves a very low value for the mean error, but it is nonetheless very expensive to be applied. All the DCP solutions are in the region below constant update policies, as they effectively bound the error and reduce the cost without needing prior knowledge the optimal inter-updates schedule.

Fig. 13 displays the number of rounds where the error made by the DCP methods go over the threshold value θ set as a parameter for the method. When compared to a model that is

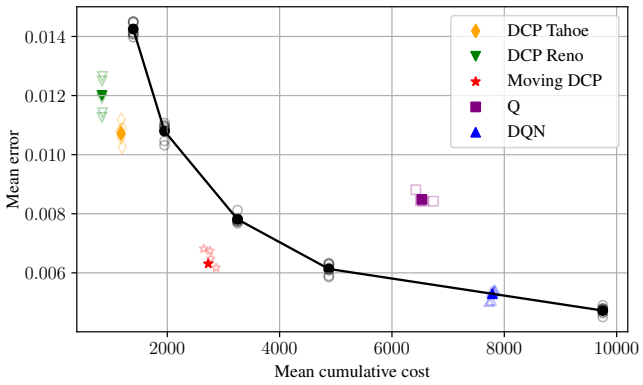

Fig. 12. Real dataset: cost-error tradeoff.

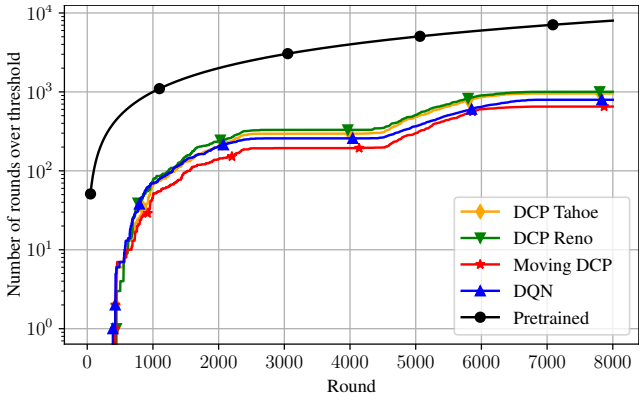

Fig. 13. Synthetic dataset: number of rounds where the error of the linear approximation is over the given threshold ($\theta = 0.03$).

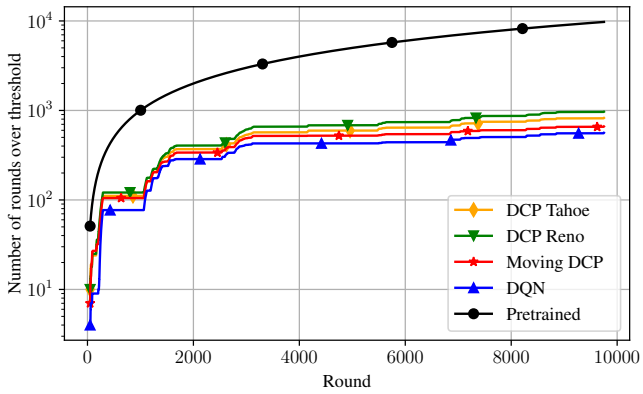

Fig. 14. Real dataset: number of rounds where the error of the linear approximation is over the given threshold ($\theta = 0.03$).

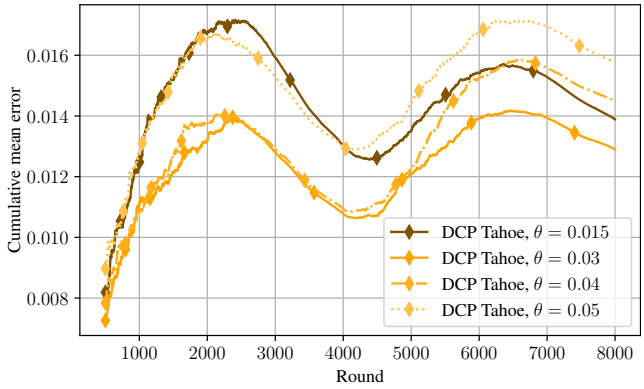

Fig. 15. Synthetic dataset: behaviour of the error's cumulative mean of DCP for different values of the model threshold parameter θ.

fully pre-trained and never updates itself according to the drift, the DCP solutions outperform it by an order of magnitude. The interesting remark is that for the most part of the experiments the methods do not perform model updates and they focus then in the sections where the drift changes more rapidly, i.e. the same regions where the mean error is growing in Fig. 5.

Fig. 14 shows a similar behaviour of Fig. 13, meaning that even in the real dataset, which has a milder but burstier drift, the proposed methods actively reduce the update frequency when it is not needed. The global saving in update requests is again of 90% when compared with an update at every round.

Fig. 15 presents the mean error for different experiments when changing the θ parameter for DCP Tahoe. By increasing the threshold value the curves increase in magnitude but they keep the same shape. This suggests that the method follows the drift but tolerates larger errors before triggering the retraining. Conversely, if the threshold value is set too low for the natural drift of the dataset the error mechanism activates much more frequently rendering the TCP inspired procedure almost useless in bounding the error effectively.

Fig. 16 displays the execution of DCP Tahoe and Reno over the synthetic dataset for different choices of the correlation factor ρ. This result shows the adaptability of our methods to different drifts as DCP Tahoe and Reno's curves for the same ρ are located in close proximity with one another and, more importantly, they show the same behaviour. Side evaluations confirmed these results when applying the two protocols in a subsampled and supersampled version of the real dataset.

V. Conclusions and Future Developments

Edge offloading can be leveraged in pervasive communication scenarios to enable simple MDs to perform complex tasks. However, communication bottlenecks prevent heavy or too frequent data exchanges to or from the ES. Our proposal is to solve this problem by having the ES performing online domain adaptation of the simple classifiers available at the MDs thanks to the richer data representation available.

Still, this opens the question of how often to update the parameters of the classifier in the presence of a dynamic domain drift. To tackle this issue, we introduced DCP, a technique that mimics the well-known approach of TCP to dynamically adjust the data exchange between the MD and the ES in the presence of errors caused by a domain drift. We proved that our proposed approach is effective in keeping the classification errors contained without clogging the communication channel, triggering domain updates only when necessary.

Possible extensions of the present work may actually take inspiration from the present approach to push the similarities with TCP further. It is well known that TCP suffers in

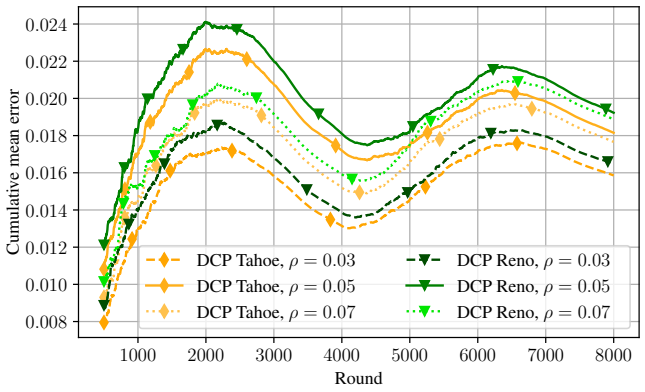

Fig. 16. Synthetic dataset: behaviour of the error's cumulative mean of DCP Tahoe and Reno for different values of the ρ parameter, that is used during the dataset construction. It is the probability that at any given round the previous point is reused, meaning that there is no drift.

wireless environment [32] to its inability of discriminating local losses due to the wireless channel from heavier losses due to congestion. To this end, some extensions for wireless TCP have been proposed. In our future work, we plan to leverage these proposals to similarly discriminate isolated losses due to errors caused by the lower complexity of the local classifiers from systematic losses due to a domain drift, which requires a more substantial intervention.

References

[1] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on Internet-of-Things devices," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4950–4960, 2020.

[2] P. Ambika, "Machine learning and deep learning algorithms on the Industrial Internet of Things (IIoT)," *Adv. Comput.*, vol. 117, no. 1, pp. 321–338, 2020.

[3] S. Paul, R. Sharma, P. Tathireddy, and R. Gutierrez-Osuna, "On-line drift compensation for continuous monitoring with arrays of cross-sensitive chemical sensors," *Sens. Act. B*, vol. 368, p. 132080, 2022.

[4] J. Bian, A. Al Arafat, H. Xiong, J. Li, L. Li, H. Chen, J. Wang, D. Dou, and Z. Guo, "Machine learning in real-time Internet of things (IoT) systems: A survey," *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8364–8386, 2022.

[5] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 2017.

[6] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comp. Surv.*, vol. 52, no. 1, pp. 1–23, 2019.

[7] D. Liu, G. Zhu, J. Zhang, and K. Huang, "Data-importance aware user scheduling for communication-efficient edge machine learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 1, pp. 265–278, 2020.

[8] Y. Qi, Y. Zhou, Y.-F. Liu, L. Liu, and Z. Pan, "Traffic-aware task offloading based on convergence of communication and sensing in vehicular edge computing," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17 762–17 777, 2021.

[9] V. Mancuso, P. Castagno, L. Badia, M. Sereno, and M. Ajmone Marsan, "Optimal allocation of tasks and price of anarchy of distributed optimization in networked computing facilities," *arXiv preprint arXiv:2404.05543*, 2024.

[10] J. Guo, S. Xia, and C. Peng, "Opa: One-predict-all for efficient deployment," in *INFOCOM*, 2023.

[11] F. Shirin Abkenar, L. Badia, and M. Levorato, "Online domain adaptive classification for mobile-to-edge computing," in *Proc. IEEE WoWMoM*, 2023, pp. 21–29.

[12] I. Burago, M. Levorato, and A. Chowdhery, "Bandwidth-aware data filtering in edge-assisted wireless sensor systems," in *Proc. IEEE SECON*, 2017.

[13] J.-A. Termöhlen, M. Klingner, L. J. Brettin, N. M. Schmidt, and T. Fingscheidt, "Continual Unsupervised Domain Adaptation for Semantic Segmentation by Online Frequency Domain Style Transfer," in *Proc. IEEE ITSC*, 2021, pp. 2881–2888.

[14] S. Zhou, L. Wang, S. Zhang, Z. Wang, and W. Zhu, "Active Gradual Domain Adaptation: Dataset and Approach," *IEEE Trans. Multimedia*, vol. 24, pp. 1210–1220, 2022.

[15] J. Zhuo, S. Wang, W. Zhang, and Q. Huang, "Deep unsupervised convolutional domain adaptation," in *Proc. ACM MM*, 2017, pp. 261–269.

[16] R. Remus, "Domain Adaptation Using Domain Similarity- and Domain Complexity-Based Instance Selection for Cross-Domain Sentiment Analysis," in *Proc. IEEE ICDM Wkshps*, 2012, pp. 717–723.

[17] M. Karimian and H. Beigy, "Concept drift handling: A domain adaptation perspective," *Exp. Syst. Applic.*, vol. 224, p. 119946, 2023.

[18] J. Moon, D. Das, and C. G. Lee, "Multi-step online unsupervised domain adaptation," in *Proc. IEEE ICASSP*, 2020, pp. 41 172–41 576.

[19] B. A. Forouzan, *TCP/IP protocol suite*. McGraw-Hill Higher Education, 2002.

[20] A. Chowdhury, S. A. Raut, and H. S. Narman, "DA-DRLS: Drift adaptive deep reinforcement learning based scheduling for IoT resource management," *Journal of Network and Computer Applications*, vol. 138, pp. 51–65, 2019.

[21] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and CO_2 measurements using statistical learning models," *En. Build.*, vol. 112, pp. 28–39, 2016.

[22] R. Wang, Z. Wu, Z. Weng, J. Chen, G.-J. Qi, and Y.-G. Jiang, "Cross-domain contrastive learning for unsupervised domain adaptation," *IEEE Trans. Multimedia*, 2023, to appear, available as early access.

[23] F. Qi, X. Yang, and C. Xu, "A unified framework for multimodal domain adaptation," in *Proc. ACM MM*, 2018, pp. 429–437.

[24] W. M. Kouw and M. Loog, "A review of domain adaptation without target labels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 3, pp. 766–785, 2019.

[25] Y. Zhang, S. Xiang, Z. Wang, X. Peng, Y. Tian, S. Duan, and J. Yan, "TDACNN: Target-domain-free domain adaptation convolutional neural network for drift compensation in gas sensors," *Sens. Act. B*, vol. 361, p. 131739, 2022.

[26] J. Tang, K.-Y. Lin, and L. Li, "Using Domain Adaptation for Incremental SVM Classification of Drift Data," *Mathematics*, vol. 10, no. 19, p. 3579, 2022.

[27] F. S. Abkenar, L. Badia, and M. Levorato, "Selective data offloading in edge computing for two-tier classification with local domain partitions," in *Proc. IEEE PerCom Workshops*, 2023, pp. 56–61.

[28] C. Li, R. Shinkuma, T. Sato, and E. Oki, "Real-time data selection and merging for 3D-image sensing network with multiple sensors," *IEEE Sensors J.*, vol. 21, no. 19, pp. 22 058–22 076, 2021.

[29] R. J. Aumann, "Borel structures for function spaces," *Illinois J. Math.*, vol. 5, no. 4, pp. 614–630, 1961.

[30] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[31] M. Shurrab, S. Singh, R. Mizouni, and H. Otrok, "IoT sensor selection for target localization: A reinforcement learning based approach," *Ad Hoc Netw.*, vol. 134, p. 102927, 2022.

[32] Y. Tian, K. Xu, and N. Ansari, "TCP in wireless environments: problems and solutions," *IEEE Commun. Mag.*, vol. 43, no. 3, pp. S27–S32, 2005.