Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

10-2019

Five challenges in cloud-enabled intelligence and control

Tarek ABDELZAHER

Yifan HAO

Kasthuri JAYARAJAH
Singapore Management University, kasthurij@smu.edu.sg

Archan MISRA
Singapore Management University, archanm@smu.edu.sg

Per SKARIN

See next page for additional authors

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Artificial Intelligence and Robotics Commons, and the Software Engineering Commons

Citation

ABDELZAHER, Tarek; HAO, Yifan; JAYARAJAH, Kasthuri; MISRA, Archan; SKARIN, Per; YAO, Shuochao; WEERAKOON MUDIYANSELAGE, Dulanga Kaveesha Weerakoon; and ARZEN, Karl-Erik. Five challenges in cloud-enabled intelligence and control. (2019). *ACM Transactions on Internet Technology*. 1-19. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/4852

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Author
Tarek ABDELZAHER, Yifan HAO, Kasthuri JAYARAJAH, Archan MISRA, Per SKARIN, Shuochao YAO, Dulanga Kaveesha Weerakoon WEERAKOON MUDIYANSELAGE, and Karl-Erik ARZEN

This journal article is available at Institutional Knowledge at Singapore Management University: https://ink.library.smu.edu.sg/sis_research/4852

Five Challenges in Cloud-Enabled Intelligence and Control

TAREK ABDELZAHER*, University of Illinois at Urbana Champaign
YIFAN HAO, University of Illinois at Urbana Champaign
KASTHURI JAYARAJAH, Singapore Management University
ARCHAN MISRA, Singapore Management University
PER SKARIN, Lund University
SHUOCHAO YAO, University of Illinois at Urbana Champaign
DULANGA WEERAKOON, Singapore Management University
KARL-ERIK ÅRZÉN, Lund University

ACM Reference format:
Tarek Abdelzaher, Yifan Hao, Kasthuri Jayarajah, Archan Misra, Per Skarin, Shuochao Yao, Dulanga Weerakoon, and Karl-Erik Årzén. 2019. Five Challenges in Cloud-Enabled Intelligence and Control. *ACM Trans. Internet Technol.* 00, 0, Article 000 (2019), 19 pages.
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

A growing category of cloud applications will be ones that support edge intelligence. These applications are brought about by a confluence of three global trends: (i) the proliferation of *connected embedded devices*, (ii) the growing need for *intelligent sensing and control* solutions to a broad set of problems, from home-automation to industrial control, and (iii) the promise of *LAN-like speeds* offered by the emerging 5G wireless infrastructure, breaking communication barriers and better supporting computational offloading. These directions will change the nature of cloud workloads, motivating new research challenges [1].

In this envisioned world, embedded devices (or "things") will be capable of human-like interactions with their environment, including speech recognition, vision, and gesture understanding. These capabilities will bring about such features as verbal device control, user authentication, and gesture-based human machine communication. Control loops will be closed by separating simple reflex-like functions that ensure safety from more computationally-involved planning, reasoning, and learning functions that will improve performance over time and offer additional assurances.

These goals will be accomplished by allowing offloading (the heavier) machine intelligence and optimization tasks, both on the sensing and control sides. Indeed, the disparity between the resource-constrained nature of embedded IoT devices and the computational needs of the aforementioned interactions suggests that data processing will be increasingly offloaded to external servers. Today, precursors of such services include speech recognition for home control chatbots (e.g., Amazon

*Authors listed alphabetically. Per Skarin and Karl-Erik Årzén contributed the most to the Journal revision.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Association for Computing Machinery.
1533-5399/2019/0-ART000 $15.00
https://doi.org/10.1145/1122445.1122456

Echo and Google Home), as well as language translation for mobile phones (e.g., SIRI and Google Assistant), done mostly in the cloud. With the increasing popularity of *edge computing*, external servers will likely move closer to the clients, and some functionality will be "cached" on the local device. On the control side, improved control strategies will be learned using computationally heavy techniques such as deep reinforcement learning. These learned techniques will set parameter values of simpler controllers that themselves could execute closer to their clients (the controlled processes).

A business, such as a management service for a shopping mall, for example, might host its own edge servers to satisfy the information processing and actuation needs of its local IoT devices. On the sensing side, these devices might include mall surveillance cameras, smart fitting rooms that suggest better-fitting items to customers, audio-based chatbots that offer directory assistance, and indeed customers' own phones (that run the appropriate app). Inference models that support such user interactions might be downloaded (after simplification to reduce size) to the individual IoT devices engaged in those interactions as a form of "caching". On the control side, the infrastructure might run reinforcement learning algorithms to optimize such actuation decisions as mall lighting, color, temperature, and background music to best complement the merchandise, encourage a more productive customer flow, and create better shopping experiences.

This paper describes key research challenges that arise in realizing the above vision. We focus on challenges specific to the introduction of *machine learning and control* as a *service* into the overall framework. The broader scope of research challenges in building a viable distributed IoT service infrastructure is clearly much more versatile. Issues, such as privacy, security, reliability, and availability must be addressed. To keep this paper focused, we limit ourselves to challenges that arise specifically from implementing the machine intelligence and control service to support future multitier IoT applications that span embedded devices, edge computing, and cloud tiers. A primary theme lies in computational offloading of key service functions while reducing aspects of service cost. We call these challenges *cloud-enabled intelligence and control* challenges.

The rest of this paper is organized as follows. To set some context, Section 2 introduces the key architectural assumptions of the discussed cloud-enabled intelligence and control services. Section 3 through Section 7 discuss the five key challenges; namely, (i) learning, (ii) quality-assured sensing, (iii) control and optimization, (iv) closed-loop guarantees, and (v) collaborative execution. Finally, we conclude in Section 8, and outline other possible future work.

2 CORE ARCHITECTURE

To set the context for the upcoming discussion of challenges, we present some broad architectural assumptions below. Consistently with common definitions of IoT, the hardware infrastructure considered here consists at the lowest level of a set of sensor and actuator devices including both low-end embedded devices such as small sensors and RFIDs, and more powerful devices such as actuators, cameras, smartphones, mobile robots, industrial robots, local control devices, and even vehicles. However, they all share the property that they have limited compute and/or storage capacity in relation to the task at hand and therefore need support for offloading. The devices are either connected directly to the Internet using cellular radio technology such as 4G/LTE or 5G or communicate with a dedicated edge node using, for example, Wifi, Zigbee or Bluetooth. The edge node can either itself participate in the offloading or serve as gateway to the Internet.

On the software side, advances in virtualization suggest an architecture where computations can be placed anywhere from an edge node or a local edge data center (e.g., one connected to a radio base station) to conventional remote data centers, and possibly even in between within the core network nodes. This software model comes in many names, including Edge, Cloudlet or Fog architecture [28]. Local data centers within the core communication network are becoming a

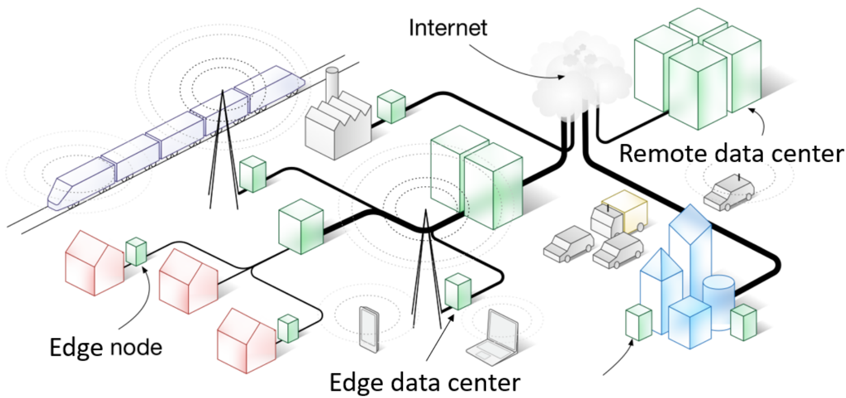

Fig. 1. System Architecture, from [2].

reality due to the network function virtualization (NFV) trend [7], where network functions (e.g., domain naming functions, caching, intrusion detection functions, and firewalls) are implemented using virtualized software and cloud technology rather than as monolithic hardware and software components. In a 5G-like framework, the timing constraints of NFVs require that these computations be located close to the wireless radio base stations. The resulting system architecture is shown in Fig. 1.

The resource characteristics of this architecture are as follows. The closer to the remote data center a computation is performed, the longer the communication delay will be, the shorter the computational delay will be (more powerful servers), and the larger the compute capacity will be (more servers). The same relationship exists for storage resources. The further up towards the remote data center, the larger the storage capacity will be. However, due to legal reasons and privacy issues, many applications have restrictions on where data may be stored. For example, some data may only be stored in a very close vicinity of the device or other data may only be stored in a node that is localized in a certain geographical region.

3 CHALLENGE I: LEARNING AS A SERVICE

How can one implement learning as a general-purpose service? The answer perhaps depends on the type of learning algorithm and the type of application. To offer more concrete examples, in this section, we focus specifically on deep learning for IoT applications with rich sensing data. In data-rich contexts, deep neural networks will likely be used increasingly as instruments of machine intelligence in the foreseeable future. This choice is motivated by the emergence of deep learning as the state-of-the-art computational intelligence solution for a large spectrum of IoT applications [26]. Besides breakthroughs in processing images and speech using deep learning techniques [8, 10], specific neural network structures have been designed to fuse multiple sensing modalities and extract temporal relationships [22]. The increasing number of studies on applying deep learning in the area of cyber-physical systems (CPS) and IoT [12, 22, 24, 27] make it a prime candidate for realizing the intelligent capabilities. Most importantly, deep learning solutions require very little customization and parameter tuning, making them convenient as a general intelligence support engine.

An interesting advantage of deep learning lies in that neural networks offer a great portable representation of learning service implementation. Much like language virtual machines (e.g., Java and Python), new services will allow the expression of processing/inference tasks in an efficient

intermediate form: This form, we argue, could be the neural network model. The model specifies network topology and edge weights, as well as other hyperparameters such as the type of activation functions used. With those parameters, it becomes possible to implement inference algorithms (specified by the model) that perform classification, prediction, estimation, or control functions. By distributing the neural network across multiple machines, it further becomes possible to distribute the service implementation.

Cloud services may then fully or partially offload from IoT devices the training and/or execution of machine learning algorithms, such as classifiers or predictors, to do a myriad of common estimation and recognition tasks based on device data such as visual inputs, speech, or gestures. Clients would ask the service to (i) generate deep neural network models (from client-supplied training data), (ii) help with (automatic) labeling of data sets, and (iii) perform model reduction (if needed for caching). Generated models might be executed as appropriate on the server, client, or any device that supports the "neural network virtual machine". System support is needed on servers to enable efficient scheduling of inference tasks (that execute the neural network models on incoming client-supplied data in real-time). A scheduler might maximize a suitably defined notion of utility to improve quality of inference results. Auxiliary functions are needed such as profiling. They will allow enhanced (neural network) model parameterization to improve accuracy and/or cost. Deep learning frameworks have at least two further advantages over alternative solutions:

- Arguably, in many scenarios, one can use laws of physics to derive the needed inference results from sensor data. For example, in a location estimation task, one can double-integrate inputs that comprise accelerometer data to obtain velocity and position. The problem with such approaches is two-fold. First, they require that application-specific models of underlying physical phenomena be developed and given to the service. Second, they rely on understanding accurate models of noise. Most estimators make assumptions on the statistical distribution of noise offering accurate results only when such assumptions are satisfied. In a complex environment, noise is hard to model. It may be non-linear, non-additive, correlated, and biased. Recent results in deep learning demonstrate that the network can learn very complex nonlinear relations, allowing better extraction of signals from noise (even when the two are intertwined in a complex nonlinear fashion) [22]. Best of all, such extraction is fully automated, thus requiring no human intervention or expertise.
- Furthermore, unlike other machine learning approaches that rely on the design of clever input features (to support the intended estimation or classification tasks), deep learning has the advantage of being able to ingest raw data directly and automatically compose relevant features by adjusting link weights. Hence, less human effort is consumed in feature engineering.

In a world dominated by data and computing devices, saving human cognitive bandwidth by employing a machine is a great trade-off. With that in-mind, we describe two key tasks that are needed to implement deep intelligence as a service. We argue that these tasks constitute the service core, although additional (more general) challenges should be solved, including privacy, security, and availability, in a typical service implementation.

3.1 Training and Data Labeling

To facilitate *learning* from data collected by the embedded devices, training and data labeling services will execute on the back-end to produce the (trained) neural networks necessary for various inference and estimation tasks. The most basic service is to ingest labeled raw data from clients and train the eventual neural network model on the server. Since it is expensive to label

a lot of data manually, another service would be to assist with *automatic* labeling. Recent work suggests the viability of automating such services:

- **Training:** In many cases, IoT devices will have already collected large amounts of sensory data (such as video footage from security cameras). Often, labels are available retrospectively (such as instances of various security breaches caught on camera). This offers opportunities for training the system to identify (and alert to) similar instances in the future. The feasibility of such a service was recently discussed in DeepSense [22], a general-purpose learning framework for sensor fusion systems. It integrates convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to extract spatio-temporal features of input signals. Sensory data are aligned and divided into time intervals for processing. For each interval, DeepSense first applies an individual CNN to each sensor data stream, encoding relevant local features. A (global) CNN is then applied to the respective outputs to model interactions among multiple sensors for effective sensor fusion. Next, an RNN is applied to extract temporal trends. Intelligent IoT applications will generally need two important functions: estimation and classification (depending on whether the sought results are continuous or categorical, respectively). Hence, at the last stage, either an affine transformation or a softmax output is used by DeepSense, depending on whether the output is an estimation or a classification result. Accordingly, it becomes possible to perform complex multi-sensor fusion tasks for purposes of estimation or classification from time-series data.
- **Labeling:** A general disadvantage of deep learning methods lies in the need for large amounts of *labelled* data. To learn well from empirical measurements, the neural network must be given a sufficient number of labelled examples from which network parameters are to be estimated. Since the number of parameters is large, so is the required number of labelled examples. In order to make deep learning services practical, a key challenge is thus to reduce the need for labeled data. One could address this challenge by employing a recently proposed approach that uses Generative Adversarial Networks (GAN) to learn from mostly unlabeled data [25]. Unlabeled data carries information on the structure of the input space. By overlaying it with labeled data, one can better observe the emergence of input data clusters corresponding to different labels. A small number of labeled points within a cluster can thus inform the labeling of the remaining points. Using this intuition, the GAN learns by playing a game of progressive refinement of both the dimensions in which points are virtually clustered and the rules for cluster separation. In this game, one entity proposes labels for unlabeled samples, whereas another tries to distinguish the resulting labeled samples from the original labeled ones. As the game proceeds, both entities learn from each other ultimately producing labels that are hard to falsify. Empirical results show that these eventual artificially produced labels (for originally unlabeled data) help improve accuracy of learning applications almost as much as the ground-truth labels themselves [25]. The approach significantly reduces reliance of the learning service on availability of large amounts of labeled data, allowing the exploitation of more easily attainable unlabeled data instead.

The limitations of automated deep learning and labeling services remain to be investigated, but preliminary evidence suggests that they are effective compared to more traditional machine learning approaches and can work well even when most data are unlabeled.

3.2 Model Reduction and Caching

Once trained, deep neural networks can be used to perform complex estimation, prediction, detection, or identification/classification tasks. Typical networks produced by deep learning techniques are very large. They may include several hundreds of layers, each composed of possibly thousands

of nodes. As such, they need to execute on appropriately well-resourced machines, resulting in communication between end-devices (e.g., sensors making new observations) and well-resourced back-ends every time the device needs to run the service on a new data item. In environments where the communication bandwidth of the end-device is not plentiful, it is advantageous to execute some inference tasks locally. This need calls for reducing the relevant neural network models to a footprint that fits the end device. Hence, a model reduction service is needed.

The feasibility of an efficient neural network model reduction service is attributed to two observations. First, it is often the case that phenomena observed by sensors evolve over lower-dimensional manifolds. In this case, the large neural network is an overkill and compression is possible. Second, in many applications, the most frequent inputs to a device comprise only a very small fraction of the much larger potential input space. For example, in a service where users typically give yes/no answers, recognizing responses such as "yes" and "no" versus neither (referring to all other utterances besides these two) should be easier than distinguishing all possible spoken words. In this scenario, neural networks produced by deep learning methods can be reduced in size without significant loss of accuracy in the common case. Much like caching, a reduced network model can run locally on the resource-limited embedded device to handle common inputs (e.g., to recognize "yes" and "no" in the above example). The identification of an uncommon occurrence (e.g., the occurrence of other words) is viewed as a cache miss that triggers full network execution on the server.

Several attempts were made to simplify deep neural networks after they have been trained. Commonly, a compression service removes edges that have low weights. The removal produces a sparse matrix (to represent the neural network), where most of the cells are zeros. The sparsity of the matrix allows for reductions in storage and computation time. Unfortunately, prior work has shown that these reductions do not scale proportionally to the fraction of zero entries in the sparse matrix [27]. This is because sparse matrix algebra is not as efficient as dense matrix algebra. Hence, as the matrix becomes sparse, additional overhead is introduced to take advantage of sparsity (compared to when it was dense), thereby offsetting some of the savings. A promising solution for a model reduction service is one that removes nodes instead of edges in the neural network to fix the above sparse matrix problem. Removal of entire nodes from the neural network is equivalent to removal of entire rows/columns from the corresponding matrix. This produces a new matrix that is also dense, but that has smaller dimensions. The approach was shown to be significantly more effective at reducing resource consumption without degrading quality [27]. The resulting compact neural network models are therefore suitable for execution on resource-limited nodes.

To automate caching, the system must decide on what constitutes frequent inference tasks. The inference models (i.e., neural networks) pertaining to those specific tasks can then be reduced and cached. Several interesting questions arise in implementing this mechanism. For example, when exactly should the system decide that an item or set of items are frequent? How small or large should the set of items be to make it worth developing a reduced model for? How to automatically adapt answers to the above two questions according to the capability of the local device, and the bandwidth of its communication link? Finally, when should the cached model be removed from the device? These questions are a topic of future work.

4 CHALLENGE II: SENSING QUALITY ASSURANCE

The complexity of sensing and control applications and the presence of learning components leads to a very important challenge: how to make guarantees on quality of results? Furthermore, in an environment where cloud-assisted execution incurs cost, it is important to understand the trade-offs between incurred resource demand and result quality. This need gives rise to several key research opportunities.

4.1 Profiling Support

Going back to the deep learning example, on the server side, execution efficiency considerations suggest the need to understand the relation between neural network structure and execution overhead. Prior work has shown that simply counting the number of neural network parameters and/or the total FLOPs involved in processing does not lead to good estimates of execution time because the relation between these predictors and execution time is highly non-linear [23]. Table 1 (reproduced from [23]) shows that neural networks with the same number of FLOPs (e.g., CNN1 and CNN2) can differ significantly in execution time. In fact, networks with fewer FLOPs can take longer to execute (e.g., CNN3 compared to CNN4).

Table 1. Execution time of convolutional layers with 3×3 kernel size, stride 1, same padding, and 224×224 input image size on the Nexus 5 phone.

	in_channel	out_channel	FLOPs	Time (ms)
CNN1	8	32	452.4 M	114.9
CNN2	32	8	452.4 M	300.2
CNN3	66	32	3732.3 M	908.3
CNN4	43	64	4863.3 M	751.7

Understanding the causes of nonlinear relations between neural network parameter settings and the resulting execution time, energy, and memory consumption is thus key to developing efficient deep learning service implementations. One may leverage recent work [23] that addressed the above challenge by implementing an automated profiling system that breaks execution models into piece-wise linear regions, and uses regression over the (automatically identified) relevant neural network parameters within each region to develop a predictive model of execution time in that region. A similar approach can be developed for modeling/minimizing energy or memory consumption. Such a profiling tool would optimize performance on the server side (as it will typically not have access to profiling results on the client). For example, leveraging the identified nonlinear behavior, it might become possible to *increase neural network size and accuracy* while at the same time *reduce its execution overhead* (as illustrated by comparing CNN4 to CNN3 in Table 1).

4.2 Result Quality Estimation

To assess the trade-off between resource cost and result quality, it remains to assess the quality of inference results produced by learning models. To support mission-critical applications, the service must offer principled uncertainty estimates that faithfully reflect the correctness of its predictions. Methods are needed that provide accurate uncertainty estimates in results obtained from deep learning models. Moreover, the uncertainty estimation must be resource efficient.

Recently, a well-calibrated and efficient uncertainty estimation algorithm was proposed for multi-sensor data fusion, called RDeepSense [24] (as an extension of DeepSense [22]). It emits a distribution estimate instead of a point estimate at the output layer. Intuitively speaking, the algorithm models node outputs with random variables and estimates their distribution parameters. Estimation of the mean of the random variable is what traditional learning does. Estimation of the variance, however, is what yields confidence in results. A smaller estimated variance corresponds to a higher confidence in the computed mean.

Interestingly, the estimation of the mean and the estimation of the variance are interrelated. Typically, the estimator jointly determines both by minimizing some error function. The choice of that function has an important effect on estimation accuracy of the two parameters. Specifically, using common error functions, such as the mean square error, were shown to underestimate the uncertainty. This is so because such an estimator predicts a very accurate mean value. If the mean

value is estimated well, the variance observed around that mean on training data is small and may thus underestimate variance encountered later during testing. In contrast, when using a nonlinear error function, such as the negative log-likelihood, the estimated mean is often biased (because the nonlinearity penalizes erring on one side more than erring on the other, causing the estimated mean to drift towards the heavily penalized side). The biased (i.e., incorrect) mean estimate results in increased measured variance around the mean, leading to an artificially inflated uncertainty estimate.

One can exploit the above intuition to arrive at an estimate of variance that neither underestimates nor overestimates the true value. The idea is to use a weighted sum of the above two error functions (namely, mean square error and negative log-likelihood) as the combined loss function [25]. The weights are adjusted (calibrated) such that the underestimation and overestimation roughly cancel out. RDeepSense was shown to generate very good uncertainty estimates that allow defining accurate confidence intervals for outputs of the deep learner.

The ability to compute confidence in deep learning results offers another interesting resource optimization possibility. Namely, one may structure a deep neural network into stages, each consisting of several layers, and compute confidence in (intermediate) results after each stage. Once a high-enough confidence is reported, it becomes possible to skip the execution of the remaining stages. For example, consider a deep neural network whose job is to identify the presence of humans in a landscape. The presence of humans may be easier to identify in some images than others. Consequently, it could be that fewer stages need to be executed for some images to reach an acceptable level of confidence in results.

4.3 Assured-Quality Run-time Inference

Putting profiling and quality estimation together, it is possible to offer an assured-quality run-time inference service. The goal is to perform inference with a required degree of quality. The service would accept data from end devices that may choose to offload inference processing to the server, and return inference results together with a confidence estimate. An important design consideration is scalability, which calls for execution efficiency. Recent studies on deep learning have shown that improvements in result accuracy diminish with increased depth of the neural network [8]. Hence, efficiency considerations suggest that once the desired quality is achieved, the service should refrain from executing additional layers.

One idea would be to schedule inference tasks in a way that optimizes total utility. The resulting overall run-time inference architecture is described in Figure 2. As shown in Figure 2, the deep neural network is separated into multiple layers. These layers are grouped into a small number of stages (of multiple layers each). At the end of each stage, a thin softmax function layer is attached to compute a classification at selected internal layers, as well as confidence in such classification. The scheduler determines how many stages to execute to avoid diminishing returns. This architecture is described in more detail in [1].

5 CHALLENGE III: OFFLOADING OPTIMIZATION AND CONTROL

Complementing the challenges described so far, that focus on intelligent (open loop) inference from sensor data, we now describe challenges in closing the loop. Much like intelligent sensing, certain control functions can be offloaded to more capable nodes. Hence, while the previous sections focused on the *neural network model* as the element of characterization (to compute its parameters), offloading, or caching, in this section we consider the *controller* to be that element. Below we explore characteristics of control models and give several examples of cooperation between client, edge, and cloud nodes for executing them.

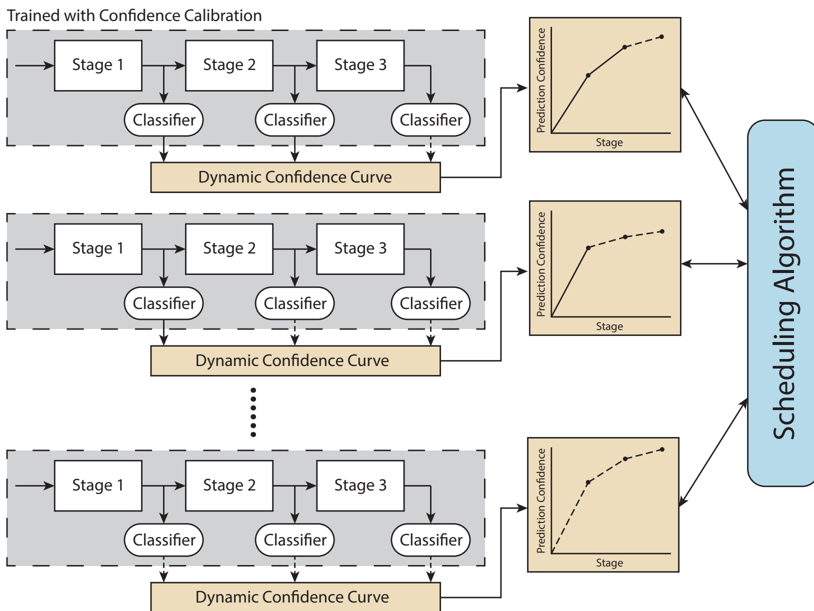

Fig. 2. An Architecture for Utility Maximization in Deep Intelligence Services.

5.1 Control Characteristics

A first question to ask is what type of control that may benefit from offloading. In industrial process control the basic control is often provided by Proportional, Integral, and Derivative (PID) controllers. These controllers require very few computations (e.g., around 15-20 lines of C code is enough for a good PID controller including the code for the logical safety network). The same holds for control in home automation, where often even simpler controllers are used (e.g., on-off controllers, proportional controllers, or integral controllers). For these types of controllers offloading is not worthwhile.

However, there are a number of control loop examples for which offloading is realistic. One such example is when a (possibly simple) controller interacts with a more complex sensor of the sort described earlier in this paper. In that category are controllers based on vision feedback. Consider a camera used as a sensor. Compute-intensive image processing may be needed. For example, in a self-driving car, a deep neural network might be used to extract object information from a video stream in order to recognize obstacles, determine positions and trajectories of other vehicles, and finally control the trajectory of the autonomous car.

Another example is optimization-based control. Here, the control algorithm consists of the on-line solution to an optimization problem. The computational demands of this optimization problem depend on both the nature of the underlying control problem and the related safety and performance requirements. The simplest case consists of a linear controlled process and an optimization problem with a quadratic cost function and linear constraints. This gives rise to a quadratic programming problem for which very efficient solvers are available. Yet, even with state-of-art solvers, the computation time may vary substantially depending on whether the constraints are active or not. In the case of nonlinear processes, either gain-scheduling between a set of linear optimization problems can be used or a nonlinear optimization problem must be formulated. The

latter can be very time consuming and require far more compute resources than what is available on local devices.

The third example is control based on models derived using machine learning. Here a number of settings are possible. One possibility is to model the controlled process using the nonlinear neural network equivalent of a auto-regression, moving aggregate (ARMA) model. The ARMA model of a linear single-input, single-output (SISO) process is given by

$$\begin{aligned} y(k) + a_1 y(k-1) + \cdots + a_n y(k-n) = \\ b_0 u(k-d) + \cdots + b_{n_b} u(k-d-n_b) \end{aligned} \tag{1}$$

where $y(k)$ is the process output at time k, $u(k)$ is the process input at time k, n is the order of the process, d is the pole excess of the process, and n_b is the number of zeros of the the process. Using a, possibly deep, recurrent neural network (RNN), the corresponding nonlinear process model can be obtained as:

$$\begin{aligned} y(k) = \mathrm{RNN}(y(k-1), \ldots, y(k-n), \\ u(k-d), \ldots, u(k-d-n_b)) \end{aligned} \tag{2}$$

This model can then be trained, for example, using stochastic gradient descent by observing true process input and output as training data. The model can also be extended to cover noise in the form of an ARMAX model (ARMA model with exogenous inputs). Another example of machine learning-based control is reinforcement learning, see Section 5.3.

5.2 Challenges and Opportunities in Controller Offloading

A general problem with offloading controllers is the increased latency from the sensing to the actuation that it incurs. Control performance and stability crucially depend on the latency and the variations in latency, also known as jitter. The longer the latency, the worse the performance. It is often possible to partially compensate for the effects of the latency, but it can never be completely undone. In general, the compensation becomes easier the smaller the jitter is.

A second problem with offloading, in particular when wireless networks are involved, is the risk of completely losing the connectivity between the local device that is connected to the process under control and the node to which the computations have been offloaded. Many control applications are mission-critical and require that the controller promptly reacts to disturbances and commands (e.g., changes in the set-point or reference signal). Therefore it is essential that the control that is offloaded and is executing remotely (e.g., at the edge or in a remote data center) is complemented with a local controller executing on the local device that is able to provide some basic level of performance in case of connectivity loss. Related to this is also the question of when to switch between the local and remote controllers, and which information the decision to switch should be based on. A problem related to this is dynamic changes in the latency characteristics caused by migration of the offloaded computations between different nodes in the network due to resource load balancing and other cloud artifacts. This means that latency will not only vary due to varying computation times (e.g., in an optimization algorithm) and communication delays (e.g., due to packet collisions), but also due to changes in the placement of the computations and to varying request admission times.

However, using the cloud for offloading also has several advantages. The illusion of infinite compute and storage resources that the cloud and the edge/fog provide opens up a number of interesting possibilities for control applications. The resources can be used for executing more advanced control strategies (e.g., based on online optimization and learning using massive data sets), than what is possible on the local device. The cloud can scale resources with the problem and implement efficient strategies for each computation. This allows the controller to evaluate complex problems that are too computationally demanding to perform locally. Information made available

through the communication network (e.g., additional more complex models and information about other similar application types) can be incorporated and used to improve the control, avoiding the overhead and potential concerns of communicating this information to the local device.

5.3 Learning-Based Control

The use of learning techniques in closed loop control has a long history. One of the first examples is *dual control* proposed in the mid 1960s [5]. The term dual refers to the need for the controller to both online identify, or estimate, the model of the process, and to utilize this model to control the process under uncertainty. This trade-off is the same as what is known in the machine learning community as the trade-off between exploration and exploitation. The optimal solution to the dual control problem can be found using value functions and dynamic programming [3]. Dual control was relatively popular in the control research community during the 1970s but because the approach suffers from the curse of dimensionality this interest soon decayed due to the lack of computing power at the time. Instead the focus shifted to adaptive control formulations based either on approximations or reformulations of the optimal dual control problem.

The interest for learning-based control has exploded during the last 10 years, mainly as a result of the success that reinforcement learning (RL) has had for various applications [21]. A major reason for this success is the availability of large-scale compute facilities based on hardware acceleration and cloud technology which is available now but not during the 1970s. RL has many similarities with dual control. Both frameworks are based on dynamic programming and for both the trade-off between exploration and exploitation is essential, see [17] for a comparison. The major successes of RL have, however, been found for applications where the state space and the action space are discrete, e.g., different game playing applications such as Atari [15], AlphaGo for playing Go [19] and AlphaZero for playing chess [18]. There the results have been spectacular, by far outperforming the best human players. However, as soon as the state and/or action spaces are continuous the results are so far less convincing. Also for discrete domain applications it is very common that the size of the state space is so large that it cannot be represented explicitly, e.g., the size of the state space of Go is $\approx 10^{170}$. Instead, the value functions are approximated using some function approximation method, e.g., a deep neural network, and then the difference between applications with discrete and continuous state-spaces becomes smaller. For example, Google and DeepMind are currently developing RL-based systems for autonomous data center cooling control [13]. Hence, a generic deep NN service that supports offloading of machine learning applications for IoT as described in Section 3 is applicable also to on-line learning-based control.

5.4 Optimization-Based Control

An increasingly popular control technique is optimization-based control. Here, an optimization-problem is solved repeatedly at each sampling instant. The optimization problem is formulated so that it minimizes some cost function subject to the dynamics of the process under control and constraints on, e.g., the control signal, the process output, or the process states. Depending on the process model and the cost function this can be more or less time-consuming. It is also typically such that the time it takes to perform the optimization varies depending on how close the constraints are to being violated, i.e., the closer the constraints are to being violated the longer time it takes to solve the optimization problem.

The most commonly used form of optimization-based control is Model-Predictive Control (MPC) [16]. In the common case it is assumed that the process is linear and that the cost function to be minimized is quadratic in the process state and control signals. This leads to a convex optimization problem.

A discrete-time linear Model Predictive Control (MPC) is specified by Equation (3). It uses the cost function $l(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k$, a cost P applied to the final state (referred to as the *terminal cost*), a system model defined by matrices A and B, and inequality and equality constraints set by the matrix vector pairs G, g and H, h respectively. The cost matrix Q penalizes moving away from the desired state while R penalizes the control signal.

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}}\ J = & \sum_{i=k}^{k+N-1} x_i^T Q x_i + u_i^T R u_i + x_{k+N}^T P x_{k+N} \\ \text{subject to} \quad & x_{i+1} = A x_i + B u_i \\ & G \begin{bmatrix} x_i \\ u_i \end{bmatrix} \le g,\ H \begin{bmatrix} x_i \\ u_i \end{bmatrix} = h \end{aligned} \tag{3}$$

The optimization is executed at each sample and the output from the optimization is a time sequence of control signals that should be applied to the process from time step $k+1$ to $k+N$, where N is the prediction horizon. However, only the first signal in the sequence is normally used and applied as an input to the process. The remaining signals are simply discarded and the entire operation is performed anew at the next sampling instant. The latter is what gives rise to the feedback property in the MPC approach. An issue that still often is approached heuristically is how to chose the prediction horizon N. A small horizon implies a smaller optimization problem that is faster to solve, but which may lead to an unstable closed loop control system and an unfeasible optimization problem, i.e., where it is not possible to find a solution that fulfills the constraints. A long horizon is advantageous with respect to stability but generates a larger optimization problem that takes longer time to solve. The prediction horizon can be viewed as a hyper-parameter for MPC.

The large computation time required for MPC compared to most other controller types makes it a natural candidate for offloading. In the next section a cloud-assisted approach to offloading of the MPC computations is presented that takes latency variations and connectivity losses into account and offers guarantees on closed-loop behavior in the presence of delays and connectivity disruptions between edge and cloud.

6 CHALLENGE IV: CLOSED LOOP GUARANTEES

A key challenge in controller offloading is to provide some guarantees on closed loop performance, even in the presence of unpredictable latency or loss of connectivity. Specifically, we shall address the problem of guaranteeing *closed loop stability*. Stability often requires that the control loop remain closed. The general solution is thus to use a local controller that can take over from the remote controller when communication is lost or the latency is too long. A challenge is to understand the conditions under which such a hybrid scheme might offer stability assurances. Beyond stability, one can also consider other guarantees, such as those on worst-case response time, maximum overshoot, or worst-case settling time.

Control theory offers rich literature on techniques used to attain the aforementioned guarantees in the context of controlling both physical [4] and computational [9] systems. Most of that literature assumes sufficient connectivity between the control algorithm on one hand, and the sensors and actuators on the other. When the controller is in the cloud, this basic connectivity assumption underlying (most) existing literature may be violated. Lack of predictable connectivity makes it challenging to offload control algorithms away from the controlled system. Below, we exemplify that challenge in the context of offloading MPC controllers.

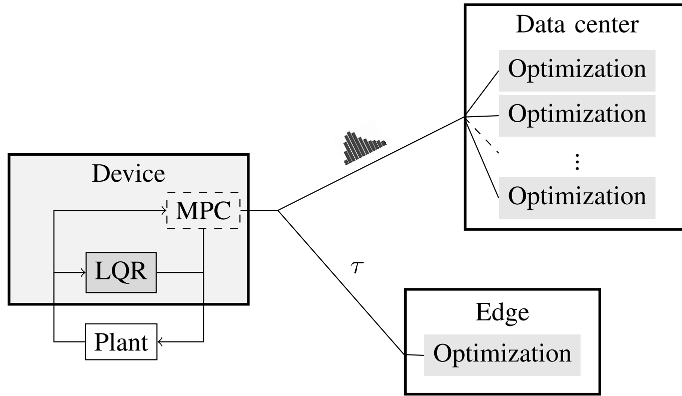

Fig. 3. Cloud assisted MPC

The local controller implemented in the device is a *Linear Quadratic Regulator* (LQR) [11]. The LQR and the MPC controllers have strong connections. For example, the LQR controller is obtained as the solution to Equation (3) when the horizon N goes to infinity and there are no constraints. However, the LQR can be derived analytically through solving a Riccati equation without requiring on-line optimization. The resulting controller is a simple state feedback controller, see Equation (4), of low complexity making it straightforward to implement it on a local resource-constrained device.

$$u_k = -Kx_k \tag{4}$$

The idea behind the cloud-assisted offloading approach described here is to, under normal and fault-free conditions, execute the MPC in the cloud. The estimated state variables are communicated from the local device to the cloud at every sampling instant and the resulting control sequence is returned to the local device where the actuation is performed. The state estimator is setup so that it at time instant k estimates the state at time $k + 1$ and uses this as the input to the MPC. When results are returned, which we here assume takes place sometime between time k and $k + 1$, the actuation is delayed until time $k + 1$. This results in a constant control latency of one sampling period, a period during which the controller continuously collects results. However, the approach can be extended to multiple sampling periods.

What is particular to this cloud-assisted MPC approach is that, due to the freedom that is available in selecting the horizon N, not only one MPC optimization request is performed in the cloud, but several, each with a different value of N. The requests could be executed at the same node or at different nodes, e.g., some could execute in an edge node and some in a remote data center, see Figure 3. Depending on where the execution takes place the communication and computation latency will be different. Some of the request responses will be lost, either because the optimization problem was unfeasible or that the latency caused the response to arrive at the local device too late, i.e., after time $k + 1$, and therefor was discarded. The result of this is that at time $k + 1$ when the actuation should be performed several control signal sequences are available corresponding to different horizons. he way the problem has been constructed all of these responses, if selected, give rise to a working MPC controller. This leads to a choice in the selection of N and the possibility to modify the placement of subsequent requests based on a trade-off between operational cost and control performance. In the next sample the procedure is repeated. With connectivity maintained this creates a closed loop MPC which is stable, feasible, and uses a variable prediction horizon.

If no responses have returned in time at time $k + 1$ then the cloud-assisted controller switches to the local LQR controller. However, if this is done instantly the controller may rapidly change course and violate constraints. An alternative would be to use an available control signal sequence generated from a request at an earlier time instant, e.g., $k - 1$. This sequence would contain $u(k+1), u(k+2), \dots u(k+N)$, i.e., it is in theory possible to use these pre-calculated control signals during the prediction horizon while still issuing MPC optimization requests every sampling instant. As a result of this new MPC responses may become available, making it possible to switch back to the nominal operating mode, or the control signal sequence would terminate and then the switch to the LQR controller would have to be performed. The drawback with this scheme is that the process would run in open-loop as long as the pre-calculated control signal sequence was used, ignoring disturbances and operator commands to, e.g., change the desired set-point of the controller. Therefore an approach is used where a weighted control law gradually shifts the dominant control signal from the one obtained from the cloud MPC to the one obtained using local LQR control, according to below,

$$u_k = \beta(k)\kappa_l(\cdot) + (1 - \beta(k))\kappa_r(\cdot),\ \beta \in [0, 1] \tag{5}$$

where $\kappa_l(\cdot)$ and $\kappa_r(\cdot)$ are the LQR and the pre-calculated MPC control laws. When the transition starts β is set to zero. When the transition is finished it is set to one, i.e., only LQR control is used. During the transition β gradually increases from zero to one. In the case there are multiple "old" control signal sequences to chose among, the one with the shortest horizon can be is used in order to switch to LQR mode as quickly as possible, thus improving the disturbance handling. During the LQR mode the local controller still issues MPC requests making it possible to switch back to the nominal mode once the network connectivity is restored.

The proposed cloud-assisted MPC control scheme is described in more detail in [20]. A benefit of this approach is that the stability conditions can be strictly imposed while allowing the horizon to be undefined. By using the cloud, the design provides flexibility and solves some of the inherent difficulties of MPC.

7 CHALLENGE V: COLLABORATIVE EXECUTION

In many environments, IoT devices are not deployed individually, but rather as a *collection* of possibly heterogeneous nodes that together support an application. In this distributed environment, how can one support such collaborative inferencing?

7.1 Distributing the Inference Model

Consider again a neural network that performs complex inferencing tasks that require substantial resources. The multistage nature of neural networks allows for an interesting possibility to share the load between clients and servers. Namely, in performing inference, it may be possible to execute some stages of the neural network on the client, leaving other stages to execute on the server. If the confidence in results obtained on the client is sufficiently high, no subsequent offloading to the server is needed. Otherwise, processing continues on the server. The approach raises questions regarding optimal partitioning of the model between the client and server. An ideal partitioning should maximally reduce client reliance on remote processing on the server, while observing client-side resource constraints as well as communication bandwidth constraints between the client and server.

An extension of this collaboration model is one where multiple distributed sensors (the clients) contribute data to be collectively used as input to the inference process. In one realization, clients would send their raw data to the server. The server would execute the entire neural network model on received data from all clients in order to compute inference results. In many cases, however, it

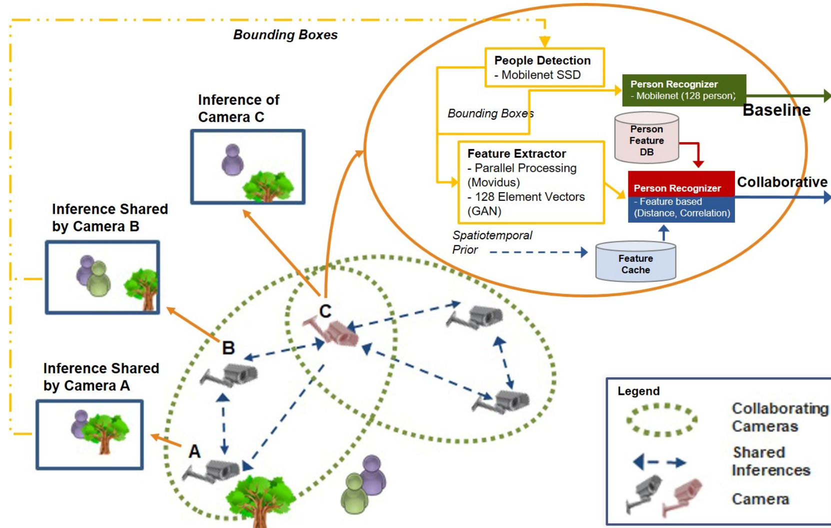

Fig. 4. Collaborative IoT (Camera) environment & Deep Inferencing Pipelines.

may be more efficient for clients to execute some part of the inference network locally on their own data then send intermediate results to the server to continue model execution remotely. In the latter case, how should the inference model be partitioned among nodes in the distributed system? Optimal partitioning can take into account resources available on individual nodes, communication bandwidth among them, as well as any end-to-end requirements such as maximum allowable latency. Viewing neural network models as the intermediate code representation for a virtual machine implies potential for great flexibility in how execution is partitioned in the distributed system. Adaptive algorithms are needed to maximally exploit this flexibility (e.g., in mobile or dynamic environments) where connectivity, power, and other local resources may change over time.

7.2 Orchestrating Collaboration

A more interesting form of cooperative processing is one where the distributed devices cooperate to mutually enhance each other's performance. For example, two cameras may realize that they are looking at the same target (e.g., because of the way they are positioned, and because of the location of the target in their respective fields of view). Hence, rather than performing target classification twice in two independent tasks, each running on inputs from one of the cameras, it might be possible to join the tasks for better accuracy. How and when should one perform such a join to best enhance classification results based on the collective data of both cameras? Note that, individually, the two cameras might not have enough information to conclude that they are observing the same target (e.g., they might not know that they have overlapping fields of view). However, a server, observing classification outputs of the two cameras over time, may conclude that their fields of view are indeed overlapping. This knowledge can thereafter be used to determine if their outputs should be processed jointly to improve accuracy of classification. The same wisdom may apply to sensors of different modalities, such as microphones and vibration sensors. In short, an edge server offering intelligence as a service for a number of IoT devices may serve the additional function of discovering correlations among their data (e.g., inferred from correlations in produced labels) that can thereafter be used to reconfigure, and possibly re-train, the neural network model to better exploit the data from these correlated sources.

Table 2. Collaborative Deep IoT Inferencing

Approach	Detection Accuracy	Recognition Latency
Individual	68%	550 msec
Collaborative	75.5%	25 msec

Consider, for example, a set of surveillance video cameras, deployed across a smart university campus (as illustrated in Figure 4) to support applications such as people *counting* (estimating the aggregated occupancy in different parts of the campus) or people *tracking* (capturing the movement trajectory of a specific individual throughout the campus). Conventionally, we can envisage that each camera operates as an *isolated* IoT device, applying state-of-the-art DNN-based techniques, such as *MobileNet* Single-Shot Detectors (SSD) [14], to perform object (people) detection, followed by object (people) identification, on each frame. Such an approach, however, has two limitations: (a) *poor processing efficiency*: executing 2 independent DNNs even on a specialized edge node (e.g., Intel's Movidius™neuromorphic co-processor) consumes $\approx$ 550 msecs/frame, implying a processing throughput $\leq$ 2fps; (b) *lower accuracy*: individual cameras may often be affected by specific context-based artifacts (e.g., occlusions, poor lighting) that impair the object detection process.

To overcome these limitations, it is possible to explore the notion of *collaborative inferencing*, where the inferencing pipelines of different IoT devices exchange *state information* in near real time and subsequently adapt their individual execution logic. As a specific illustrative example, consider Figure 4, where each camera has a field-of-view (*FoV*) with varying degrees of overlap with neighboring camerasâĂŞe.g., cameras B and C both observe two individuals and a tree (from different perspectives) concurrently. In this scenario, the cameras may collaborate to improve their overall operational efficiency and accuracy. For example, one camera that detects individual bounding boxes (individuals) in its FoV may share those bounding box coordinates with its neighboring cameras. The other peer cameras can then supplement their own DNN-based inferences with these additional object coordinates (suitably remapped to a common coordinate space) to improve both their detection accuracy (for people counting) and reduce their processing latency (for individual tracking).

The collaborative paradigm described above was evaluated with the PETS dataset [6], consisting of 8 outdoor cameras. Table 2 summarizes the performance differences between the baseline (non-collaborative) vs. the collaborative deep inferencing approach. We see that such collaboration is indeed beneficial: it increases the people counting accuracy by $\geq$8%, and achieves a 20-fold reduction in the average per-frame processing latency.

7.3 Services for Collaborative Inferencing

To realize the benefits of such collaborative deep inferencing, we believe that it will be important to provide several new forms of functionality. These include:

- *Collaboration Brokering:* The collaborative video monitoring example provided earlier implicitly assumes that the cameras are aware of each other's identity & the extent of FoV overlap. Note that, such overlap need not be concurrent: one can envisage future scenarios where the camera views are temporally correlated with a variable lag. For example, two corridors at two ends of a campus building corridor are likely to observe the same individuals 20 seconds apart. To easily support such dense IoT deployments, it is necessary to discover such correlations, and establish the identity of collaborators, in a more autonomous fashion. By operating on the metadata and higher-level inferences from individual nodes, it becomes possible to discover and establish the relevant collaboration parameters. For example, a broker node

may instruct cameras A & B to apply the collaborative tracking mechanism discussed above, but with a time lag of 20 seconds. Developing suitable mechanisms that uncover such useful spatiotemporal correlations among IoT devices, while satisfying the requirements of low communication overheads and privacy, is an open challenge.

- *Resilient Collaboration:* Collaborative deep inferencing, however, introduces a new form of failure: their operation is vulnerable to incorrect or malicious behavior by individual IoT nodes. For example, false or noisy bounding box estimates by one camera can reduce the people detection accuracy of other peer cameras by over 20%. To promote practical use of such collaboration paradigms, the architecture must also provide *resiliency services* that offer protection against such adversarial behavior. One may need to continuously monitor the output inference streams, and the internal parameters of relevant deep pipelines, of individual IoT devices to first (a) proactively uncover faulty operational situations and subsequently (b) provide suitable pipeline modifications to compensate for such faults.

8 CONCLUSIONS

This paper described five challenges in endowing future IoT applications with cloud-assisted machine intelligence. Several service components were presented together with related challenges at both training and inference time. Both sensing and control applications were considered. The work aims to produce early prototypes of machine intelligence services for IoT systems, and contribute to the realization of a new smart edge, where each device appears endowed with unlimited knowledge and intelligent behavior. Indeed, understanding the true potential, capabilities, and limitations of cloud-assisted intelligence and control may be the first step towards revolutionizing our interactions with physical surroundings in the near future.

It should be understood that many other challenges exist in exploiting cloud computing to support future IoT applications in general. Different types of applications may have different emphasis, requirements, customers, and hence different challenges. The five challenges described in this paper deal more directly with computational offloading to support intelligence and control as a service, distilled from the authors' own hand-on experience with related research projects. While limited in scope, the work hopefully makes a step forward towards understanding the broader eventual challenge landscape.

ACKNOWLEDGMENTS

This material is supported partially by the National Research Foundation, Prime Ministers Office, Singapore, under its International Research Centers in Singapore Funding Initiative. It was also partially supported by the Nordforsk University Network HI2OT, Sweden, and by the ITEA3 project AutoDC. The research was also sponsored in part by NSF under grants CNS 16-18627 and CNS 13-20209, in part by the US Army Research Laboratory under Cooperative Agreements W911NF-09-2-0053 and W911NF-17-2-0196, and in part by WASP (Wallenberg AI, Autonomous Systems and Software Program) funded by the Knut and Alice Wallenberg foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsors, the Army Research Laboratory, NSF, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] Tarek Abdelzaher, Shuochao Yao, Yifan Hao, Yiran Zhao, Ailing Piao, Huajie Shao, Dongxin Liu, Shengzhong Liu, Shaohan Hu, Dulanga Weerakoon, Kasthuri Jayarajah, and Archan Misra. 2019. Eugene: Towards Deep Intelligence as a Service. In *In Proc. 39th IEEE International Conference on Distributed Computing Systems (ICDCS).*

[2] Karl-Erik Årzén, Per Skarin, William Tärneberg, and Maria Kihl. 2018. Control over the edge cloud – an MPC example. In *1st International Workshop on Trustworthy and Real-Time Edge Computing for Cyber-Physical Systems, Nashville, USA*.

[3] Richard Bellman. 1957. *Dynamic Programming* (1 ed.). Princeton University Press, Princeton, NJ, USA.

[4] Richard C Dorf and Robert H Bishop. 2011. *Modern control systems*. Pearson.

[5] A.A. Feldbaum. 1965. *Optimal Control Systems*. Academic Press. https://books.google.se/books?id=_dEXtQEACAAJ

[6] James Ferryman and Ali Shahrokni. 2009. Pets2009: Dataset and challenge. In *2009 Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*. IEEE.

[7] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. 2015. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine* 53, 2 (Feb 2015), 90–97. https://doi.org/10.1109/MCOM.2015.7045396

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

[9] Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. 2004. *Feedback control of computing systems*. Wiley Online Library.

[10] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[11] R. E. Kalman. 1960. Contributions to the theory of optimal control. *Boletin de la Sociedad Matematica Mexicana* 5, 2 (1960), 102–119.

[12] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 283–294.

[13] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. 2018. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*. 3814–3823.

[14] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *ECCV*.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[16] J.B. Rawlings and D.Q. Mayne. 2009. *Model Predictive Control: Theory and Design*. Nob Hill Pub.

[17] Benjamin Recht. 2018. A Tour of Reinforcement Learning: The View from Continuous Control. *CoRR* abs/1806.09460 (2018). arXiv:1806.09460 http://arxiv.org/abs/1806.09460

[18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144. https://doi.org/10.1126/science.aar6404 arXiv:https://science.sciencemag.org/content/362/6419/1140.full.pdf

[19] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (Oct. 2017), 354–. http://dx.doi.org/10.1038/nature24270

[20] Per Skarin, Johan Eker, Maria Kihl, and Karl-Erik Årzén. 2019. An assisting Model Predictive Controller approach to Control over the Cloud. *arXiv e-prints*, Article arXiv:1905.06305 (May 2019), arXiv:1905.06305 pages. arXiv:cs.SY/1905.06305

[21] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[22] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. 2017. DeepSense: a Unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee.

[23] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek Abdelzaher. 2018. FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys '18)*. ACM, New York, NY, USA, 278–291. https://doi.org/10.1145/3274783.3274840

[24] Shuochao Yao, Yiran Zhao, Huajie Shao, Aston Zhang, Chao Zhang, Shen Li, and Tarek Abdelzaher. 2018. RDeepSense: Reliable Deep Mobile Computing Models with Uncertainty Estimations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 4 (2018), 173.

[25] Shuochao Yao, Yiran Zhao, Huajie Shao, Chao Zhang, Aston Zhang, Shaohan Hu, Dongxin Liu, Shengzhong Liu, Lu Su, and Tarek Abdelzaher. 2018. SenseGAN: Enabling Deep Learning for Internet of Things with a Semi-Supervised Framework. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3, Article 144 (Sept. 2018), 21 pages. https:

//doi.org/10.1145/3264954

[26] Shuochao Yao, Yiran Zhao, Aston Zhang, Shaohan Hu, Huajie Shao, Chao Zhang, Su Lu, and Tarek Abdelzaher. 2018. Deep Learning for the Internet of Things. *Computer* 51, 5 (May 2018), 32–41. https://doi.org/10.1109/MC.2018.2381131

[27] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM.

[28] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. 2019. All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey. *Journal of Systems Architecture* (February 2019).