Cloud-Hosted Intelligence for Real-time IoT Applications

Ken Birman, Bharath Hariharan, Christopher De Sa
Computer Science, Cornell University. {kpb3, bh497, cmd353}@cs.cornell.edu

May 31, 2019

Abstract

Deploying machine learning into IoT cloud settings will require an evolution of the cloud infrastructure. In this white paper, we justify this assertion and identify new capabilities needed for real-time intelligent systems. We also outline our initial efforts to create a new edge architecture more suitable for ML. Although the work is still underway, several components exist, and we review them. We then point to open technical problems that will need to be solved as we progress further in this direction.

1 Introduction

Today's datacenters employ a wide variety of distributed services to support forms of *edge computing*. In the popular press, edge computing often refers to computing that occurs directly on the sensors, but we have a different concept in mind: a model in which intelligent tasks are hosted in the cloud data center, but triggered by events initiated from outside the data center. Moreover, our intelligent cloud subsystems are presumed to interact with the sensors, configuring them in a continuous and highly dynamic control loop. This vision of an edge partnership changes the computational roles of the external devices: a camera might pivot, focus, change its image sensing parameters and so forth, but wouldn't try to understand the image. The cloud brings intelligence to bear yet also leverages the unique capabilities of the sensors.

One should understand cloud infrastructures as a shared environment, in which each "ecosystem" has specialized roles, tools and services, but shares standard elements. For example, the developer of a web-page-oriented application finds existing three-tier infrastructures and tools that cover most elements of most web development tasks, and will guide the creation of an efficient solution, and the similarly for a developer of a database app, a streaming media platform, a mobility solution, etc. However, we lack such an ecosystem for the new kind of edge computing envisioned here. In today's prevailing cloud infrastructure, machine learning occurs in batched back-end infrastructures (such as MapReduce), while real-time actions occur as close to the edge as possible. The delay between acquiring new knowledge and starting to use it at the edge can be substantial. Moreover, we are not aware of any situation in which a back-end system dynamically reconfigures an edge sensor under time pressure.

The puzzle is that nobody wants to discard the powerful existing technology base. Thus the challenge is to reuse standard cloud technology when possible, but in a very different environment. Further considerations center on cost: today's back-end machine learning infrastructure was shaped by costs, and its pervasive batching of updates reflects trade-offs between overhead and scalability. In the edge, where learning would sometimes occur in response to individual events, treating each event as a batch of size one would be ineffective. This, in turn, suggests that for some tasks, new kinds of incremental machine learning algorithms will be key.

2 Real-time machine learning

To illustrate the new capabilities that will be required by truly real-time machine learning, consider a somewhat contrived concrete example. Suppose that Alice wishes to trap and relocate a skunk that has nested somewhere on her property, while not wanting to disrupt her neighbor's cats, which also roam her property. Her plan is to track the movements of the skunk with a network of cameras and then place several baited IoT-enabled intelligent (animal-friendly!) traps, which will close when the skunk is known to be inside. The novelty is that Alice's cameras and traps will also recognize the cats, closing the traps to safeguard the bait and avoid trapping the wrong animals.

On the one hand, cats look similar to skunks,[1] so accurately distinguishing between the two requires computationally heavy machine learning models that cannot be run on the meager computational resources of the cameras. Alice's plan is to improve the quality of classification by combining data across cameras. However, the resulting models will be computationally intensive and will therefore need to run at least partially in the cloud. Additionally, they will need to be responsive in real-time so

[1] As famously illustrated in the *Pepé Le Pew* cartoon animations.

that the traps can decide whether to remain open (if the skunk is nearby) or close (if non-skunk animals are nearby or if the skunk is caught).

We now encounter some of the many new elements of the problem. A first issue is that the cameras probably will not be able to do much of the heavy lifting: they will be power and bandwidth constrained, with limited computational capabilities. On the other hand, Alice probably wants to leverage the features they *d*o support, and those could include parameterized preprocessing of value to her ML pipeline. For example, video cameras might be able to auto-focus, might support dynamic range control, could have infrared or other unusual spectral options, and might even have hardware able to help detect when an animal is present in the frame. Video cameras also have substantial storage: data is initially captured into on-camera storage, and Alice will want to configure it to only report thumbnails. Then she can implement an intelligent algorithm to decide which videos to actually download and process. The video-acquisition "event" would thus trigger Alice's cloud-hosted logic, and her logic would decide whether or not to initiate a download, determine whether to close the trap or keep it open, etc.

Recent progress in computer vision can enable these kinds of classification tasks and support the associated models [3, 5]. Even so, Alice's system would be difficult to build on top of present IoT and cloud-edge technologies. Current IoT infrastructures are designed mostly to employ models acquired in the past, and any learning involves saving data into files for batch-processing outside of the real-time event path. As a result, the existing platforms do not offer a suitable place to host the kind of low-latency ML model learning that Alice's IoT application will need.

Worse, when Alice's application does initiate a download, the ensuing pipeline might require rapid but heavy processing to perform tasks such as coalescing multiple events from different devices into a single event for unified processing, even when those events all represent the same real-world occurrence (e.g. multiple camera views of the skunk). In today's systems, such tasks lean heavily on hardware accelerators such as GPU, TPUs and FPGA clusters. But for cost reasons, these are deployed primarily at the back end, where they can be assigned to one long-running task at a time and then used to process an entire batch of new data. This approach makes sense in existing systems, and amortizes overheads, but it requires a tolerance of substantial processing delays.

As Alice works to move this sort of functionality to the edge, she will encounter a number of puzzles. One is that the edge is "opinionated" about how one interfaces devices like video cameras to the cloud. In the prevailing model, the devices themselves are managed by some form of hub, and it reports on new events by triggering lightweight *functions*, which run within elastic multi-tenant servers. A function, in this terminology, is just a simple program coded in a standard language, but functions are small, run in containers, and can be launched in as little as a few milliseconds. They would normally compute very briefly, after which the container is garbage-collected. Thus, this entire model is mismatched with Alice's needs!

There is another edge option, albeit one that has not received as much attention. Today's IoT edge is supported by heavier vendor-supplied *services*, which are used by the function layer in many ways. Some just cache data that functions can rapidly load through a key-value API. Others perform tasks like image compression and still others provide message bus or message queue functionality, whereby a function can dispatch work to some kind of back-end layer. One could imagine augmenting the handful of existing services with a new tier of user-created μ-services, consisting of one or a few long-running processes hosted on more powerful servers with attached accelerators. Such approaches are common for web applications, but the needs of intelligent real-time applications are sufficiently different so that creating a μ-service capability for the reactive edge poses interesting research challenges.

If this were solved, Alice could implement a two-tier solution: she would create a set of customized intelligent functions to handle events as rapidly as possible, but then use a cloud "app-service" to deploy a collection of μ-services. These services would construct the real-time knowledge models her applications will need, perform real-time classification and data fusion, and carry out other tasks with real-time or online intelligence roles. Because the μ-services will be long-running, any initial setup delays would off the critical path that determines real-time responsiveness.

Thus, we could solve the platform technology gap by offering Alice a set of tools to enable this kind of application development. But she will then be faced with doing some fundamental research on what might be categorized as *real-time machine learning.*

To see this, consider the ML side of Alice's application. An always-on ML system cannot know *a priori* of all possible scenarios it will encounter in deployment. For example, if a hitherto-unseen animal, such as raccoon or possum enters Alice's property, she'll need to be careful or the very dumb preprocessing occurring in the video cameras themselves might dismiss the unfamiliar shape as noise. Conversely, Alice wouldn't want to trap these animals, given that her only real goal is to capture the skunk so that she can move it to a skunk paradise far from her home. In this single example we see a surprising number of relatively unexplored challenges in machine learning (sometimes called open world recognition and never-

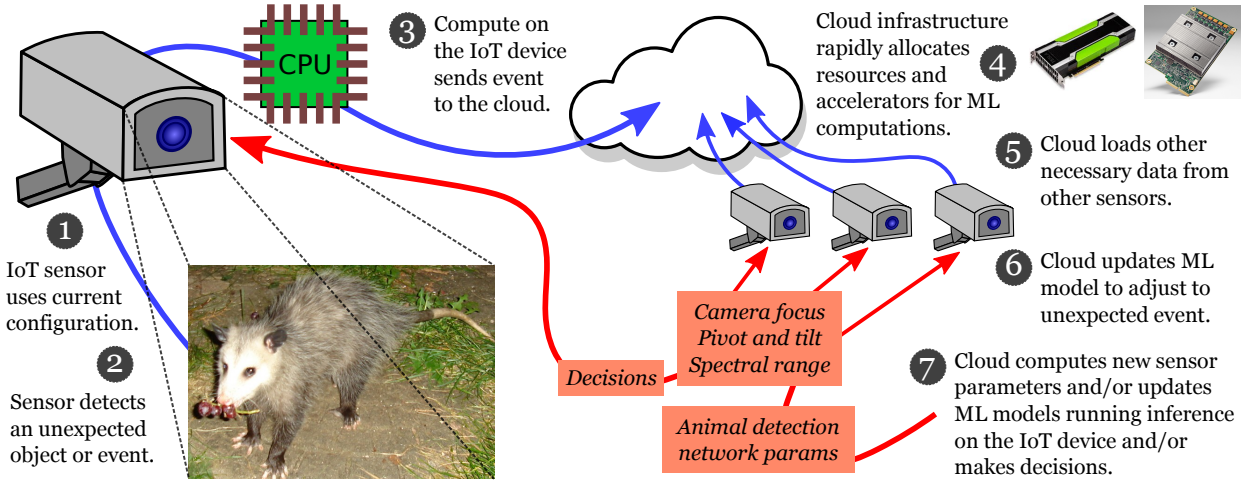

Figure 1: Illustration of the proposed capabilities of intelligent real-time IoT.

ending learning [4]). These problems that require significant interactions between the cloud edge and the IoT devices. For example, if Alice's cameras are modestly intelligent, it would be plausible that one could upload a trained convolutional neural network model for image classification. The cameras would then be able to identify interesting images that should be uploaded for further analysis on the cloud. Viewed as a whole, the cloud edge might need to (1) identify that it has observed anomalous behavior, (2) query the cameras for their recent video data; (3) add this new kind of animal to its recognition vocabulary, (4) re-train all Alice's cameras' local classifiers to now classify the opossum as interesting; and (5) push these changes to the cameras, so that they now generate IoT events for the opossum. And all this needs to be done in real-time, before the possum gets the chance to eat all the bait in all the traps. And all of it would need to be fault-tolerant, scalable and self-managed. This overall need is for what we call *intelligent real-time IoT*. To enable it we will need to build both new ML technologies and new IoT infrastructure, while reusing the same cloud-hosted μ-service management frameworks that work so well for today's web applications and three-tier database solutions.

3 Bridging the gap

To see how this gap can be bridged, lets first drill down on the existing IoT infrastructure. Today's cloud support for IoT centers on security: sensors are managed by a specialized IoT hub responsible for securely binding to the devices, managing software version levels, and pushing any required configuration parameters. Customization entails writing functions that will be triggered by IoT events and can carry meta-data (sensor data, thumbnails, etc). The functions would then either handle the event, or forward it to μ-services capable of the heavier processing tasks required for visual analysis or intelligent planning.

But now consider the way that existing ML technologies are implemented: generally speaking, they are not designed for long-term active learning in a small pool of edge-based servers, even ones with the computer power to run a language like TensorFlow [1] and with attached accelerators (such as FPGAs, GPUs and TPUs [14]). In our view, support for these accelerators is not minor: the importance of using them will only grow as specialized architectures for fast inference continue to be developed [12,17]. Indeed, beyond computational accelerators, high performance will almost certainly require selective deployment of high-speed non-volatile memories, and because we will want to react in real-time as new videos are captured, the use of remote direct memory (RDMA) hardware to move the bytes. Since copying within the operating system can actually be far slower than RDMA, a zero-copy style of computing will also be needed.

We thus are confronted with challenges that could require compiler extensions (to eliminate copying, at least for large objects), operating system extensions (to ensure that these large objects will be placed in parts of memory properly mapped and registered for RDMA transfers), RDMA-enabled tools for data handling (for example, to deliver multi- perspective video data to a GPU configured to fuse them into a 3-D movement model), and out-of-band control mechanisms for interacting dynamically with the parameters of the cameras. We might need our video transfers to bypass host memory entirely, sending the data directly from the edge sensors into an accelerator to avoid the delays of first staging it through host memory. And beyond all of these issues are questions of fault-tolerance, consistency, security, and privacy.

4 Cloud/Device cooperation

Of great interest is the idea that the host and the sensor or actuators could work together as an intelligent, autonomous, real-time "solution." Some tasks can be performed directly on cameras, but the devices would often have to be parameterized with recently acquired models prepared on the cloud edge. Conversely, some tasks that arise in the vision system cannot occur in the cameras (such as multi-perspective data fusion), and can only be carried out by sending those data fragments to the cloud, where we can set up a processing network consisting of functions and μ-services tailored specifically for that role.

The ML developer today lacks abstractions to describe this online style of intentional learning, in which the system not only learns models, but then might need to split the learned model between the "device portion" and the "cloud portion." To enact such a split, we would also need platform abstractions to facilitate the needed interactions, reconfiguring the sensors to change their behavior on the fly as part of a single real-time knowledge cycle.

Our data fusion example illustrates all aspects of this puzzle, and the problem will also arise in systems of much greater societal importance than skunk-relocation. For example, consider the challenge of continuously updating a model of traffic on a highway using a stream of snippets of video (or videos, showing the same highway lanes from multiple cameras from multiple angles). Just as Alice's "novel animal" scenario exposed problems that no single camera could perform, here we encounter related needs, but at a much more ambitious scale. The current highway context would itself be a very large and dynamic ML model. Tracking such a model and performing the integration of the multi-perspective data could only occur in the cloud (a term that can include point-of-presence clusters that situate portions of cloud functionality physically close to some set of sensors).

But even if we cannot perform the desired tasks on the IoT sensors themselves, it is equally infeasible to simply send all of this data to the cloud and process everything off-camera. First, even if cameras have limited capabilities, those capabilities are still powerful, and we may not be able to recover the lost information if we ignore the on-camera processing step. Additionally, a genuinely large IoT sensor deployment could capture immense amounts of completely useless imagery and video from vast numbers of redundant perspectives, and shipping all of this information to the cloud only to discard it is likely to be prohibitive both due to the limited bandwidth of edge IoT network links, and because of the limited power of the IoT devices themselves.

To address this, ML algorithms running on the cloud edge will need to (1) select a subset of the event-associated data that they need, and have ways to request that the IoT devices send those data, and (2) offload *some* storage and compute tasks to the IoT device[2], but in a selective manner, and optimized to make use of the limited device storage and compute capacity. The cloud might even anticipate a kind of data of interest, and move quickly to reposition a camera or some other sensor to capture that data: a tight real-time constraint. Current IoT infrastructures do not support any of these modalities.

Our conclusion is that much as the big-data frameworks at the back have evolved, the same must occur on the edge: we will need a new kind of systems platform, and it will enable the creation of an exciting new generation of real-time ML solutions. In the big data, batch processing environments, evolution was driven by success on high-value use cases, and we have seen a steady growth in the sizes of the data that need to be processed. This created the context within which today's machine-learning tools, specialized hardware and batch-driven execution environments were created [16]. Tomorrow, we would argue, a similarly rapid revolution will transform data-intensive ML applications aimed at reactive real-time settings.

5 Preliminary steps

In our work at Cornell, we are starting to make inroads on this agenda.

Systems side. The Derecho platform [13] is a new C++ library that we created to assist developers of intelligent μ-services[3], and designed to integrate easily into the prevailing IoT computing architecture. With an eye towards the new reactive intelligent edge, we are now tackling the challenge of using Derecho as a component of the processing pipeline just described. Derecho already integrates efficiently with user-provided logic, but because machine-learning algorithms are often expressed in ways that facilitate the use of existing libraries, we are starting to explore efficient support for important computational motifs such as MapReduce and Stochastic Gradient Descent that can be expressed using MXNet, a popular C++ package offering a wide variety of ML kernels.

Every step of this undertaking it turning out to be challenging. When building Derecho itself, we made an early decision that we would support a virtually synchronous variant of Paxos, configurable either as an in-memory multicast or as a persistent data logging mechanism, with time-indexed versioning. But as we set out to create this protocol and map it to RDMA, we encountered a puzzle: RDMA performance is so extremely high that it requires a whole new programming style, in which protocols must

[2]For example, a machine learning algorithm could perform dimensionality reduction on the IoT device to compress the data before sending it to the cloud.

[3]`https://github.com/Derecho-Project/`

be split into a control plane and a data plane [15], then further modified to be as asynchronous as possible, so as to keep the RDMA hardware continuously busy. Even common functionality such as multithreading with lock-based protection can create delays that significantly impede performance at these data rates. Eventually, we found it necessary to split Derecho's data update path completely away from its query path, so that neither can disrupt the other. This was successful, and the current Derecho library can fully load our RDMA network, while read-only computation chases the update stream, accessing stable data a few microseconds behind the update frontier.

To fully leverage these features in a complex ML application, we will need to consider a number of dimensions. The application itself probably needs to be coded in C++, avoid unnecessary copying (which is far slower than RDMA data movement), minimize locking, and shift unnecessary work off the runtime critical path. Beyond those steps, one might want to restructure the ML algorithm itself so that its time-sensitive patterns of data flow will match the sweet spot for RDMA data movement.

Our initial target, MXNet, is well matched to these goals: MXNet is coded in C++, and the developers who created it already had a goal of minimizing synchronization and copying on the critical paths. Even so, it isn't trivial to arrange for MXNet kernels to directly access data held within Derecho's key-value versioned object store, and vice versa. Beyond this, the use of a greater variety of hardware accelerators pose challenges. Derecho itself focuses primarily on RDMA and NVM, and on the efficient mapping of Derecho's critical path to the hardware's fastest modes of use. If RDMA is available, Derecho moves bytes entirely via direct DMA transfers from machine to machine, at data rates that can reach 16GB/s on our current generation of 100GB Mellanox switches, both on Infiniband and on RoCE. If NVM is available, Derecho can leverage the highly efficient SPDK interfaces to offer persistence via direct memory writes.

But as we integrate ML algorithms that run over MXNet with Derecho, we might want Derecho's data replication paths to support the use of FPGA as a bump in the wire (or perhaps bump in a group of wires, if we are replicating data over a cluster). We might want those data paths to terminate directly in a GPU or TPU cluster, without "staging" all data through host memory first (which entails an extra DMA transfer). Given that the raw data can be held on the sensors for a period of time, we might want to consider forms of fault-tolerance that would require recomputing certain data, rather than simply storing one or more replicas of the raw data, or of computed artifacts. Moreover, as seen above our longer vision is to focus on a computing model that integrates lightweight stateless functions with a collection of new μ-services hosted in an app-service environment. Developers such as Alice will need to find it easy to use the system to create these intelligent services, easy to test and deploy the solutions, and also easy to configure them to leverage hardware acceleration as appropriate. Today, none of these steps would be trivial.

ML research. We are also beginning to make progress on the ML side of the agenda. Traditional machine learning systems, especially in complex sensing modalities such as vision, must be trained offline on large labeled training sets. This precludes rapid adaptation of the model to new unseen situations in the real time settings discussed above. Such rapid adaptation requires that the adaptation be inexpensive both in terms of computation and in terms of training data, since it is unlikely that large amounts of labeled data can be collected in real time.

Unfortunately, it is unclear how the deep learning models that power modern systems can be trained under such data and computation constraints. In our recent work we have begun to build deep learning models that can incrementally learn new classes from limited training data without much computation. In terms of techniques, our work belongs to a class of approaches called *meta-learning*, which use simulated experiments to train learning machines that take training sets as input and produce trained models as output [9, 18, 19]. However, in a departure from this paradigm, instead of just searching for better learning machinery, our key insight is to design learning machines that leverage *additional information* to make up for limited training data. This might be through modeling common modes of intra-class variation so that limited training sets can be automatically "augmented" [10, 20], or by taking into account additional annotations that might be available, such as provided or inferred figure-ground information [21]. At the same time, we have taken care to design these systems so that they fit real-world data and computational budgets but are effective in the challenging problems encountered in practice. This approach has led to significant performance gains on practical benchmarks: for example, in our most recent work, we *doubled* the performance of state-of-the-art models [21].

While these results are promising, they do not address power and communication constraints that are characteristic of IoT systems. To address these challenges, additional techniques are needed for training with a limited communication bandwidth between the device and the cloud and training networks that support low-power inference. In some of our recent work, we have developed algorithms that support compressed-communication learning with theoretical guarantees [2]—such guarantees could be used to eventually build a system that automatically compresses communication without the user (i.e. Alice) needing to worry about the statistical effects. We have also developed methods for fast and efficient learning and

inference with limited computational resources [22, 23], including parallel and low-precision learning [7]. This work is part of a broader line of research into algorithms for efficient learning and inference being done in the machine learning systems community [16], but the specialization of these techniques to the IoT domain remains mostly unexplored.

But there is still a large gap between incremental models trained quickly on small amounts of training data and large, offline models. Quick adaptation and learning requires models to have a fairly in-depth, generalizable understanding of the input domain, something that modern machine learning models still fail to develop. Future work will require building models that discover and learn aspects of the input domain beyond the provided labels (such as the 3D structure of objects), so that these can be used as intermediate representations for faster learning. More generally, however, we will need to make fast, real-time adaptation a *first-class citizen* in the design and evaluation of learning models, as opposed to just the fixed test-set performance that we focus on today.

One possible way to make real-time adaptation a first-class citizen is to use *Bayesian machine learning* methods that model the time-varying nature of reality with a time-varying statistical model such as a Markov chain or other type of graphical model. These approaches can avoid the difficulties with adapting models based on a fixed test set by baking the real-time nature of the task into the machine learning model. To make such a model work in real time, we will need algorithms to run inference and learning on graphical models in real time, which will allow us to support rapid decision making that takes time into account at all scales. In the IoT space, this is presently a challenge because existing real-time machine learning systems are focused almost entirely on inference for fixed models trained *a priori*, and cannot be directly applied to Bayesian ML which reasons on a higher level about distributions over models. Some of our prior work makes steps towards this task by applying classic optimization techniques, such as minibatching [6], parallelism [8], and streaming memory access orders [11] to graphical model tasks. But more work will be needed to make these powerful Bayesian methods fully robust and real-time.

6 Conclusion

Machine learning technologies provide an exciting opportunity to improve the utility of IoT devices. But at present, there is a mismatch between the abstractions used in ML and in IoT systems that makes it more difficult to both deploy ML in the IoT space and develop new ML capabilities for IoT tasks. The overall need is for new abstractions on both sides that can support what we call intelligent real-time IoT, an ecosystem that can spin up heavyweight processing in real-time to actively update an ML model that can make real-time decisions about IoT events or actions. We described our approach to bridge the gap between ML and IoT technologies, and showed how our preliminary steps at Cornell are starting to make progress on this agenda.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: A system for large-scale machine learning.

[2] Acharya, J., De Sa, C., Foster, D. J., and Sridharan, K. Distributed learning with sublinear communication. *arXiv preprint arXiv:1902.11259* (2019).

[3] Beery, S., Van Horn, G., and Perona, P. Recognition in terra incognita. In *The European Conference on Computer Vision (ECCV)* (September 2018).

[4] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E. R., and Mitchell, T. M. Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010).

[5] Cui, Y., Song, Y., Sun, C., Howard, A., and Belongie, S. Large scale fine-grained categorization and domain-specific transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2018).

[6] De Sa, C., Chen, V., and Wong, W. Minibatch Gibbs sampling on large graphical models. *arXiv preprint arXiv:1806.06086* (2018).

[7] De Sa, C., Feldman, M., Ré, C., and Olukotun, K. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *ACM SIGARCH Computer Architecture News* (2017), vol. 45, ACM, pp. 561–574.

[8] De Sa, C., Olukotun, K., and Ré, C. Ensuring rapid mixing and low bias for asynchronous Gibbs sampling. In *JMLR workshop and conference proceedings* (2016), vol. 48, NIH Public Access, p. 1567.

[9] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML* (2017).

[10] Hariharan, B., and Girshick, R. Low-shot visual recognition by shrinking and hallucinating features. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 3018–3027.

[11] He, B. D., De Sa, C. M., Mitliagkas, I., and Ré, C. Scan order in Gibbs sampling: Models in which it matters and bounds on how much. In *Advances in neural information processing systems* (2016), pp. 1–9.

[12] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[13] Jha, S., Behrens, J., Gkountouvas, T., Milano, M., Song, W., Tremel, E., Renesse, R. V., Zink, S., and Birman, K. P. Derecho: Fast State Machine Replication for Cloud Services. *ACM Trans. Comput. Syst. 36*, 2 (Apr. 2019), 4:1–4:49.

[14] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., luc Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. In-datacenter performance analysis of a Tensor Processing Unit. In *International Symposium on Computer Architecture (ISCA)* (2017).

[15] Peter, S., Li, J., Zhang, I., Ports, D. R. K., Woos, D., Krishnamurthy, A., Anderson, T., and Roscoe, T. Arrakis: The operating system is the control plane. *ACM Trans. Comput. Syst. 33*, 4 (Nov. 2015), 11:1–11:30.

[16] Ratner, A., Alistarh, D., Alonso, G., Andersen, D. G., Bailis, P., Bird, S., Carlini, N., Catanzaro, B., Chayes, J., Chung, E., Dally, B., De Sa, C., Dean, J., Dhillon, I. S., Dimakis, A., Dubey, P., Elkan, C., Fursin, G., Ganger, G. R., Getoor, L., Gibbons, P. B., Gibson, G. A., Gonzalez, J. E., Gottschlich, J., Han, S., Hazelwood, K., Huang, F., Jaggi, M., Jamieson, K., Jordan, M. I., Joshi, G., Khalaf, R., Knight, J., Konečný, J., Kraska, T., Kumar, A., Kyrillidis, A., Lakshmiratan, A., Li, J., Madden, S., McMahan, H. B., Meijer, E., Mitliagkas, I., Monga, R., Murray, D., Olukotun, K., Papailiopoulos, D., Pekhimenko, G., Ré, C., Rekatsinas, T., Rostamizadeh, A., Sedghi, H., Sen, S., Smith, V., Smola, A., Song, D., Sparks, E., Stoica, I., Sze, V., Udell, M., Vanschoren, J., Venkataraman, S., Vinayak, R., Weimer, M., Wilson, A. G., Xing, E., Zaharia, M., Zhang, C., and Talwalkar, A. SysML: The new frontier of machine learning systems. `https://www.sysml.cc/doc/sysml-whitepaper.pdf`, 2019.

[17] Sandler, M. B., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[18] Snell, J., Swersky, K., and Zemel, R. S. Prototypical networks for few-shot learning. In *NIPS* (2017).

[19] Vinyals, O., Blundell, C., Lillicrap, T., kavukcuoglu, k., and Wierstra, D. Matching networks for one shot learning. In *NIPS*. 2016.

[20] Wang, Y.-X., Girshick, R., Herbert, M., and Hariharan, B. Low-shot learning from imaginary data. In *Computer Vision and Pattern Recognition (CVPR)* (2018).

[21] Wertheimer, D., and Hariharan, B. Few-shot learning with localization in realistic settings. In *Computer Vision and Pattern Recognition (CVPR)* (2019).

[22] Zhao, R., Hu, Y., Dotzel, J., De Sa, C., and Zhang, Z. Building efficient deep neural networks with unitary group convolutions. *arXiv preprint arXiv:1811.07755* (2018).

[23] Zhao, R., Hu, Y., Dotzel, J., De Sa, C., and Zhang, Z. Improving neural network quantization without retraining using outlier channel splitting. *CoRR abs/1901.09504* (2019).