Energy, Latency and Staleness Tradeoffs in AI-Driven IoT

Naveen T.R. Babu and Christopher Stewart, The Ohio State University

ABSTRACT

AI-driven Internet of Things (IoT) use AI inference to characterize data harvested from IoT sensors. Together, AI inference and IoT support smart buildings, smart cities and autonomous vehicles. However, AI inference consumes precious energy, drains batteries and shortens IoT lifetimes. Deep sleep modes on IoT processors can save energy during long, uninterrupted idle periods. When AI software is updated frequently, scheduling policies must choose between interrupting deep sleep and degrading AI inference by delaying updates. Scheduling is challenging because of the stochastic nature of update arrivals, processing needs and updates cannot be delayed indefinitely. This paper studies scheduling policies when (1) updates (tasks) arrive frequently, (2) updates must be processed within staleness limits and (3) energy footprint is the metric of merit. We define a scheduling policy as a sequence of choices that decide when updates are applied. We use random walks to explore the space of scheduling policies and $2^K r$ design of experiments to quantify primary effects and interactions between factors. We conducted 6 $2^K r$ tests with 5X replication each. Each test executes 1,000,000 random walks and computes their energy footprint. We simulated multiple IoT, e.g., varying the number of AI inference components from 5–500. The best random-walk policy uses much less energy than 99^{th} and 95^{th} percentiles. First-come-first-serve and shortest-job-first policies use 7X more energy than the best policy.

ACM Reference Format:
Naveen T.R. Babu and Christopher Stewart, The Ohio State University. 2019. Energy, Latency and Staleness Tradeoffs in AI-Driven IoT. In *The Fourth ACM/IEEE Symposium on Edge Computing (SEC 2019), November 7–9, 2019, Arlington, VA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3318216.3363381

SEC 2019, November 7–9, 2019, Arlington, VA, USA

ACM ISBN 978-1-4503-6733-2/19/11…$15.00
https://doi.org/10.1145/3318216.3363381

1 INTRODUCTION

Internet of Things (IoT) equips appliances, tools and vehicles with sensors, a unique identifier and a network connection. These devices can detect failures, push diagnostics to the cloud and enable cost saving analytics worth $235 billion [3]. Often, devices can be fixed, improved or adapted locally. These local solutions are delayed by moving diagnostic data to the cloud. Increasingly, IoT devices use AI software to classify problems and apply appropriate local solutions. In five years, the number of IoT devices powered by AI software is expected to triple [15].

AI software, e.g., neural networks [7] and random forests, accurately classify IoT sensor readings. They train detailed models from large, heterogeneous data sets and often classify data correctly in new contexts. Training requires power hungry compute resources that exceed the capacity of IoT batteries. Thus, cloud data centers are widely used to train AI software. After training, AI inference, i.e., using models to classify incoming readings, uses much less power. AI Inference, i.e AI models which execute Inference tasks, can be pushed to IoT/edge devices.

AI inference can be updated to capture new data distributions. For example, consider a drone set up to count corn stalks [1, 2]. Early in the growing season, training data classifies small plants as corn stalks, but as the canopy grows training data is more likely to classify small plants as unwanted weeds. As another example, consider IoT devices with limited network bandwidth. Cloud data centers can push small versions of AI inference, trained on small data sets, and later progressively push larger versions. Such progressive sampling trades classification accuracy for latency.

As AI inference proliferates, updates across many inference components accumulate large energy footprints. When updates are available, IoT schedulers have two choices: install now or delay. It helps to delay updates because, if they are installed in batches, IoT processors can enter deep sleep modes more often. This saves energy, allowing battery powered IoT to execute longer. However, delaying updates degrades answer quality (i.e., the accuracy of classifications). Staleness corresponds to answer quality [5, 6]. Hard limits on staleness can prevent large quality loss. Even with limits on staleness, stochastic arrival times and processing needs for updates make energy-efficient scheduling a challenge.

We define a batching policy as the collection of choices made by an IoT scheduler, covering every update for each

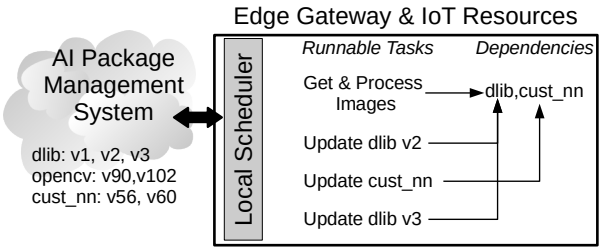

Figure 1: Workloads on AI-driven IoT include sensing, AI inference and installing updates

AI inference executing on IoT. In this paper, we first systematically sample batching policies and compare their energy efficiency. Given a trace of AI inference updates and their arrival times, we use random walks to capture a wide range of batching policies. That is, our simulator decides stochastically whether to install now or delay. We compare roughly 1M random walk policies. These results allow us to characterize variation between the best and high ranking policies. Second, we study the effects of energy needs per update, update frequency and staleness. We use $2^K r$ design of experiments to quantify primary effects and interactions between factors. We conducted 6 $2^K r$ tests with 5X replication each.

The remainder of the paper is organized as follows: Section 2 describes the software architecture and scheduling challenge facing AI-Driven IoT. Section 3 outlines the experimental framework pursued. Section 3.1 describes our simulation framework that randomly samples batching policies. Section 4 presents preliminary results, Section 5 presents related work and Section 6 presents conclusions.

2 AI-DRIVEN IOT

Figure 1 depicts update processing in AI-driven IoT. The scheduler on the IoT device manages multiple processes, including: (1) sensors, (2) multiple AI inference models and (3) package management software. Package management software queries cloud repositories to discover the most recent version of AI inference models executing on the IoT. When the version number in the cloud exceeds the local version number, the local package management software asks to install an update. Figure 1 shows sensor reading and AI inference as the primary task. There are also three outstanding updates (2 for DLib and 1 for a custom CNN).

As time progresses, the alteration in factors like changing environment factors, modification of goal and shifting data distributions degrade inference accuracy. Accuracy can be improved by retraining the AI model in cloud. The IoT device needs to update to a newer (higher) version of the model. The newer AI package is pushed from cloud to IoT. The

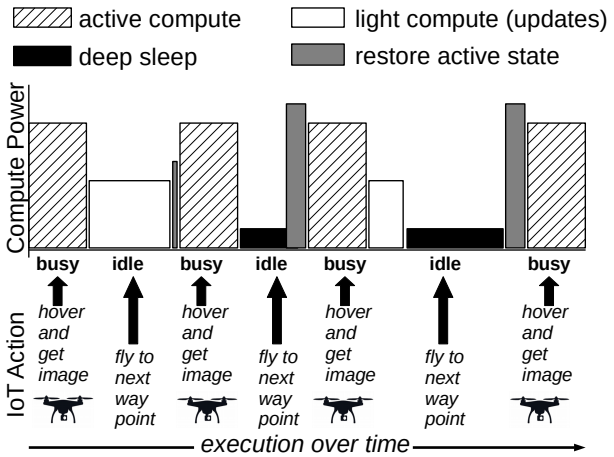

Figure 2: Batching updates saves energy

IoT scheduler runs tasks for (1) capturing images, processing images and (2) updating the AI inference. The local scheduler must decide when to install AI inference updates.

Staleness is the difference between the version of software executing on IoT and the most up-to-date version in the cloud. A staleness bound defines the maximum staleness allowed for AI software. Staleness bounds greater than zero, allow schedulers to delay updates. However, inter arrival times between updates are stochastic. Schedulers can not predict their exact arrival nor their processing needs.

2.1 Scheduling challenge:

Figure 2 depicts update scheduling, deep sleep and workload demands for a realistic IoT context: autonomous unmanned aerial vehicle [2]. Active compute refers to the CPU speed when executing sensing and AI inference workloads. To be sure, we expect CPU to execute multiple AI inference models, so the active compute state may amortize multiple energy efficient states. In Figure 2 the IoT drone hovers during this stage [2]. After the active compute period, the aircraft must fly to the next waypoint (this time-period is shown as idle period in Figure 2). During this period, the IoT device could (1) perform AI inference component update and transition to the light compute phase or (2) delay updates and transition to deep sleep. Compute power to restore active state from deep sleep is greater than restoring from light compute state. It is interesting to note that in addition to exclusive light compute and deep sleep state, the scheduler can choose to perform one or more updates and transition to deep sleep.

Figure 2 highlights the scheduling challenge for AI-driven IoT. In the first idle period, the scheduler configures the CPU for light compute and processes an update. In the second period, the CPU enters deep sleep but it is interrupted by

sensing and AI inference workloads. The CPU consumes more energy by entering deep sleep too aggressively. In the third idle period, the CPU first processes some updates but then transitions into a long deep sleep. Here, the deep sleep saves energy if there are no other outstanding updates.

It is an interesting scheduling problem to perform the scheduling in most energy efficient way. The complexity of solving this scheduling problem is NP-Hard. In designing scheduler for this problem, this paper considers traditional operating system scheduling techniques like First-come-first-served (FCFS) [16], Shortest-job-first (SJF) [16], Least-recently-used (LRU) [16]. There might be multiple factors which help in designing an efficient scheduler. These factors are discussed in Section 3.

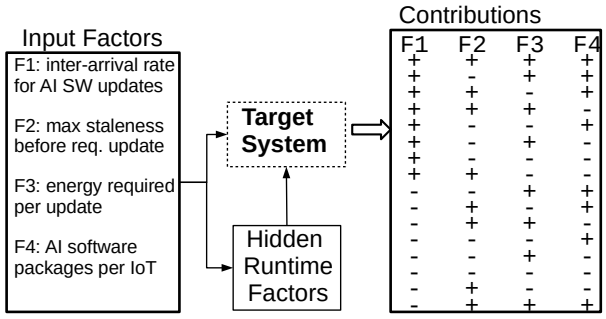

Figure 3: Design of experiments

3 DESIGN OF EXPERIMENTS

Performance analysis of a system involves measurement, simulation and analytical modeling [4]. Performance of a system might be affected by many factors. For instance, consider measuring performance of a computer. Factors are the variables like CPU type, memory size, number of disk drives that affect the response variable. Levels are the values that a factor can assume, for instance the CPU type can assume three levels such as 8080, Intel i5, Intel i7. Replication involves repetition of all or some of the experiments. Design of experiments takes into account the number of experiments, factor level and number of replications for each experiment. The goal of design of experiments should be to get maximum information by running few experiments [4].

$2^K r$ is an interesting experimental design for a study that is in very early stage. It is easy to analyze and will help us to understand the impact of different factors. In $2^K r$ design of experiments, we choose 'k' factors each will have two levels and repeat the experiments 'r' times to draw meaningful mathematical model, calculate experimental errors and minimize the loss. Sign table is an effective method to perform $2^K r$ design of experiments. In this method, each factor is represented in a column. '+' and '-' indicates the two level. We construct a sign table which is a permutation of levels for all the factors listed and conduct experiments referring this table [4].

In our study, we use $2^K r$ design of experiments to understand and characterize the space of scheduling policies. There are various factors present in the design and implementation of AI-driven IoT system. Factors like update rate, number of inference components, staleness, AI model compression technique used and power efficient architecture impact designing and characterizing an ideal scheduler. An ideal scheduler learns all these factors and adapts to changes in these factors. Clearly, this is a drawback of traditional scheduling techniques like FCFS [16], SJF [16] and LRU [16] as they do not consider these factors while scheduling updates.

Figure 3 shows the different factors and construction of sign table to run experiments on target system. We select four factors: 1. inter-arrival rate for AI inference updates with two levels, "-" as low update rate and "+" as high update rate 2. maximum staleness "K" allowed before processing the updates with two levels, "-" as K and "+" as K+1 3. energy required for update with two levels, "-" as current power saving in deep sleep and "+" as additional 2X power saving in deep sleep 4. AI inference components per IoT with two levels, "-" indicates 'x' components and "+" indicates '10x' components. These factors are important run-time factors which impact scheduling updates on a target system. In total, we conducted 6 2^K experiments with 5 times replication i.e 6 * 16 * 5 = 480 experiments. The results of execution of these experiments are found in Section 4.

3.1 Sampling update batching policies

In order to understand the energy difference between different scheduling policies, we sample different update batching schedules using random walk technique [9] across the trace of update arrivals. Given a trace of update arrival and idle time period, random walk across these traces yield different update scheduling policies. Figure 4 explains schedules from eager update policy (FCFS) and random walk across update trace. X-axis represents idle time periods and Y-axis represents different update ID (i.e. model numbers). "A" denotes new update for update ID available and "V" represents void saying there's no new update available. At the end of time-period 18, both eager update and random walk lead to same update ID version, but they differ in the way they perform updates and total energy incurred for updates. We observe that eager update policy aggressively schedules updates of newly arrived AI inference components and incurs energy close to capacity during idle period. On the other hand, random walk can generate an update schedule which

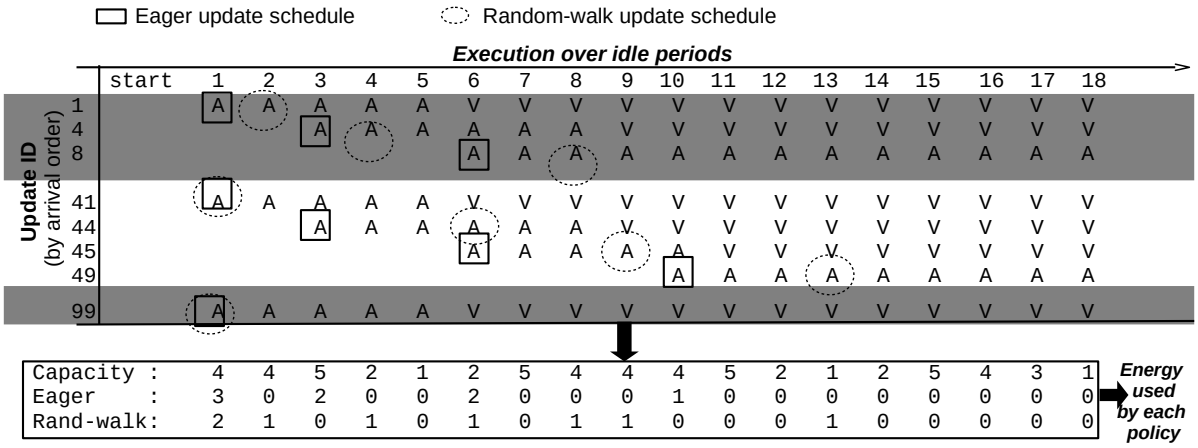

Figure 4: Sampling update batching

performs fewer updates per idle period and seek to transition to deep_sleep, in turn consuming much less energy than capacity. We noticed that random walk across the trace to fetch all the update scheduling policies would lead to very huge number of schedules and it's impractical to fetch all the schedules from a very long trace. This number increases exponentially as we scale number of models and also when we extend traces to be collected for large time period.

To overcome this problem, we propose our novel skip_ahead method. Instead of performing time consuming, computationally intensive random walk; skip_ahead covers a subset of the schedules covered by random walk. The schedules collected using skip_ahead differ from each other and capture different ways to perform update schedule.The working of skip_ahead(m,n) is as follows: perform 'm' updates ('m' is randomly chosen from {0:5}) in current idle period and skip updating (i.e device will be in deep_sleep) in the next 'n' idle period. Skip_ahead avoids performing schedules with nearly same update policy. Instead it covers the space of different schedules with variance in scheduling policy and total energy for update. We claim that this subset is sufficient to understand different ways to schedule updates. We randomly perform skip_ahead multiple times through our trace to generate schedules with different update batching policy. This method is comparable to random walk: performing skip_ahead randomly generates multiple update schedules which are distinct from each other and incur different total energy. If a particular skip_ahead cannot be performed because a model requires update due to staleness violation, we allow such must update models to be updated.

We perform different set of skip_ahead in order to collect schedules which differ from each other. Skip_ahead(m,n) by uniform n (n=0:15), Skip_ahead(m,n) randomly bounded by 0:n (n=1:15), threshold based skip_ahead (where we choose to update/skip_ahead based on the number of updates aggregated), wavy skip_ahead (where we increase the skip_ahead till a constant, reset to zero and resume increasing skip_ahead) were some of the techniques used to sample update batching. These update batching yield different scheduling policies which differ in decisions being made.

We run the simulation for about 1,000,000 iterations and store all the update batching policies. We repeat the simulation by varying different factors quoted in Figure 3.

4 RESULTS

We simulated traces of AI inference component updates using energy, storage and update time metrics from [8]. The inter-arrival time for idle and busy time-period was sampled from Poisson distribution. With traces for AI inference component updates arrival, inter-arrival time, update energy and update time;we simulate traditional scheduling methods like (FCFS) [16], (SJF) [16] and (LRU) [16]. In each of these scheduling technique we make sure staleness is being satisfied, i.e in an inter-arrival period we update all models whose version will be outdated. In order to find potentially-heuristic solution, we performed random walks [9] across update traces. We store all the results produced by these random walks and record the best, 99th percentile, 95th percentile and 50th percentile energy values.

We used 2 servers with Intel Xeon, 8 core, 32GB RAM, 2.10 GHz processor to run simulations on traces of AI inference component updates. We conducted different experiments as specified in Section 3 to understand the impact of features.

Figure 5 shows the results of performing random walks. We ran simulations for a month to understand the impact of different factors like update rate, staleness and improvements in architecture on energy required to perform updates. We varied the number of AI Inference components used for

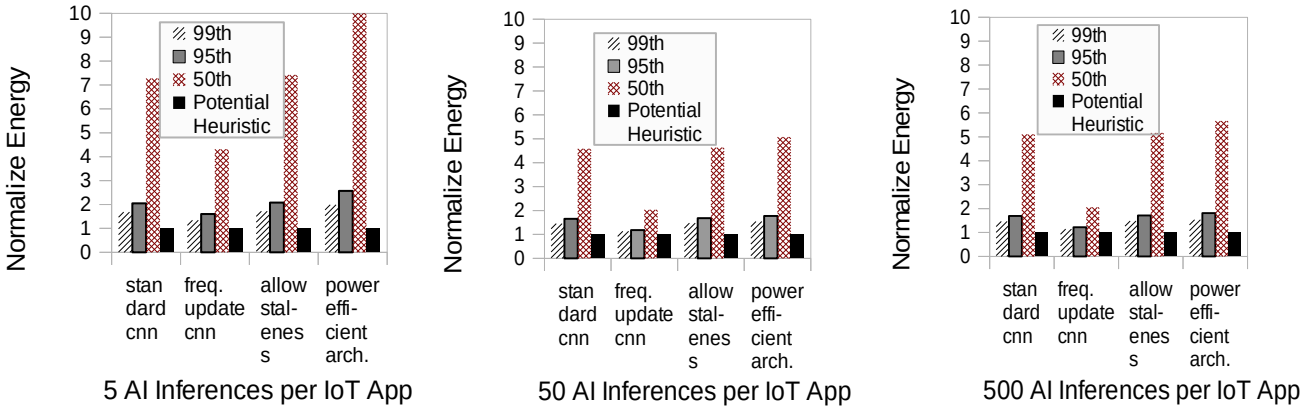

Figure 5: Early results from design of experiments

simulation from 5(left), 50(middle) and 500(right). We speculate currently 5 inference components used in IoT App, 50 inference components in next 2-3 years and 500 inference components in 10 years. Standard cnn is the baseline trace of inference update arrivals with standard update rate, staleness=2 and standard power saving when device transitions to deep sleep. Potential-heuristics is the update schedule which consumes the lowest energy. All the schedules and energy from random walks are computed and sorted to get the 99th, 95th and 50th percentile schedules.

Standard cnn (for 5 AI Inferences per IoT App) shows 99th percentile consumes 1.7X more energy than the best and 95th percentile consumes 2.1X more energy than the best. The median is 7X the best schedule. The gap between best, 95th percentile and median decreases when you increase the update rate. Increasing staleness adds relaxation to performing updates, thereby decreasing energy of every schedule by a constant amount. Using a power-efficient architecture reduces the energy consumed in deep_sleep, reducing the energy of schedules often transitioning to deep_sleep. Median scheduling policies do not transition to deep_sleep often and hence do not get benefit of energy saving using power-efficient architecture. All the three graphs (Figure 5) depict similar trend for these chosen factors.

Clearly, there's a huge gap between potential-heuristics, 99th and 95th percentile schedule. You can expend approximately 2X energy if you operate in 99th or 95th percentile compared to potential-heuristics based approach. This energy value can be 4x-10x if the update scheduling policy lies in 50th percentile. We observe that number of inference components, update rate, staleness and power efficient architecture are crucial factors when applying a scheduling policy for AI-driven IoT.

Figure 6 shows the results of scheduling AI inference components updates using traditional scheduling techniques like

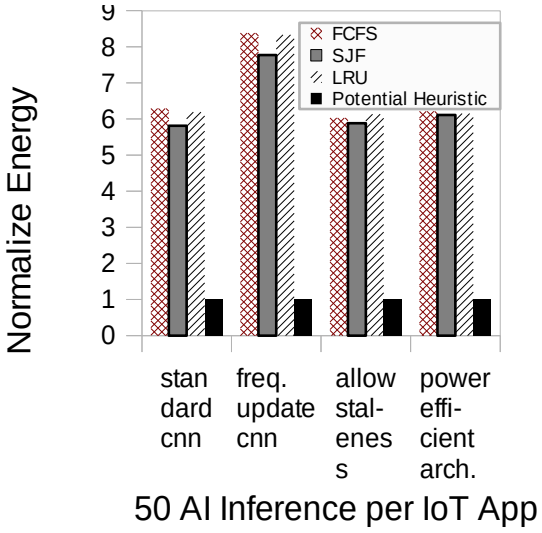

Figure 6: Traditional scheduling techniques

FCFS [16], SJF [16] and LRU [16] compared to potential-heuristic schedule. The simulation assumes 50 AI inference components per IoT application. Potential-heuristic based schedule involves performing random walks for a given trace of AI inference updates and choosing the schedule with lowest energy. Standard cnn is the baseline trace of inference update arrivals with standard update rate, staleness=2 and standard power saving when device transitions to deep sleep. We make following changes to understand factors impacting different scheduling policies: the classifier update rate is increased 5x in frequent update cnn, staleness is increased by one in allow staleness and achieve 2x power savings in deep sleep using power efficient architecture. Clearly all the experiments show that FCFS [16], SJF [16] and LRU [16]

consumes 6-7X more energy compared to potential-heuristic schedule and behaves similar to the 50th percentile schedules from random walk. This clearly shows inefficiency of traditional scheduling techniques and the need to use an online scheduling technique for practical AI-driven IoT devices.

5 RELATED WORK

PowerNap reduces power consumption when servers idle by carefully designing servers for (1) minimal energy consumption when idle and (2) minimal transition time between normal and idle modes [10]. Computational sprinting [12, 14] extend this approach to mobile and IoT devices which also save energy when idle. Our work shows that scheduling domain for such systems is challenging. Liu et al. studied the selection of deep neural network (DNN) compression techniques [8]. Such a framework selects an optimal compressed DNN model which meets user specified performance goals and resource constraints like accuracy, storage, computational cost, latency and energy consumption. Our work of scheduling AI inference updates (1) converts these models into streaming traces and (2) shows that it is challenging switch between compression techniques online. Song et al introduces their work of using autonomous and incremental deep learning for IoT systems [17]. Their work avoids significant data movement from IoT device to cloud, instead focuses on using emerging IoT devices and AI for running inference based tasks at the edge.

6 DISCUSSION

Currently, Artificial Intelligence (AI) in edge computing is still in rudimentary stage and there's enormous research going on in this area [11]. With advancement in compute power of IoT devices, there is growing interest in applying AI to edge computing [13]. In today's world, AI in edge systems aren't used to their fullest extent [11]. We believe as the use of accelerators in edge devices become common, there would be abundant AI inference components used in edge devices and these components receive frequent cloud-edge updates over period of time [17].

Traditional scheduling techniques like FCFS [16], SJF [16] and LRU [16] applied to AI inference component updates aren't energy efficient. To understand the energy difference of different scheduling policies, we performed random walks across traces of AI inference component update arrivals. Such traces showed huge gap (6X-7X) between traditional and potential-heuristic scheduling techniques.

Among all the schedules from random walk [9], we observed significant gap between potential-heuristic, 99th and 95th percentile scheduling policies. This difference worsens 6x-7x for median scheduling policy. The update scheduling problem formulation represents a challenging offline problem (NP-Hard). In practice, the IoT scheduler must make choices online, making the problem even harder.

While these results are early, there is a need for designing energy efficient online scheduler that performs like 99th or 95th percentile schedules. In future, we plan to perform an in-depth design of experiments, simulate AI-driven IoT using FAAS [1] and design Online scheduler for AI-driven IoT.

Acknowledgments: This work was funded in part by NSF Grants 1749501 and 1350941 with support from NSF CENTRA collaborations (grant 1550126).

REFERENCES

[1] Jayson Boubin, Naveen T.R. Babu, John Chumley Christopher Stewart, and Shiqi Zhang. Managing edge resources for fully autonomous aerial systems. In *ACM Symposium on Edge Computing*, 2019.

[2] Jayson Boubin, John Chumley, Christopher Stewart, and Sami Khanal. Autonomic computing challenges in fully autonomous precision agriculture. In *IEEE ICAC*, 2019.

[3] Louis Columbus. Iot market predicted to double by 2021, reaching $520b. forbes.com, 2018.

[4] Raj Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley, 1991.

[5] Jaimie Kelley, Christopher Stewart, Nathaniel Morris, Devesh Tiwari, Yuxiong He, and Sameh Elnikety. Measuring and managing answer quality for online data-intensive services. In *2015 IEEE International Conference on Autonomic Computing*, 2015.

[6] Jaimie Kelley, Christopher Stewart, Nathaniel Morris, Devesh Tiwari, Yuxiong He, and Sameh Elnikety. Obtaining and managing answer quality for online data-intensive services. In *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2017.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553), 2015.

[8] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *ACM MobiSys*, 2018.

[9] Lovasz and Laszlo. Random walks on graphs: A survey, combinatorics, paul erdos is eighty. *Bolyai Soc. Math. Stud.*, 2, 01 1993.

[10] David Meisner, Brian T. Gold, and Thomas Wenisch. Powernap: Eliminating server idle power. In *ACM ASPLOS*, 03 2009.

[11] Carlos Melendez. Ai on the edge: Is it ready for prime time?, 2019.

[12] Nathaniel Morris, Christopher Stewart, Lydia Chen, Robert Birke, and Jaimie Kelley:. Model-driven computational sprinting. In *ACM European Conference on Computer Systems*, 2018.

[13] Janakiram MSV. How ai accelerators are changing the face of edge computing, 2019.

[14] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P Pipe, Thomas F Wenisch, and Milo MK Martin. Computational sprinting. In *IEEE international symposium on high-performance comp architecture*, 2012.

[15] Report Code: TC 7047. Ai in iot market. https://www.marketsandmarkets.com/, 2019.

[16] Abraham Silberschatz, Greg Gagne, and Peter B Galvin. *Operating system concepts*. Wiley, 2018.

[17] M. Song, K. Zhong, J. Zhang, Y. Hu, D. Liu, W. Zhang, J. Wang, and T. Li. In-situ ai: Towards autonomous and incremental deep learning for iot systems. In *IEEE HPCA*, 2018.