EdgeWise: Energy-Efficient CNN Computation on Edge Devices under Stochastic Communication Delays

MEHDI GHASEMI, Arizona State University, USA
DALER RAKHMATOV, University of Victoria, Canada
CAROLE-JEAN WU, Arizona State University, USA
SARMA VRUDHULA, Arizona State University, USA

This paper presents a framework to enable the energy-efficient execution of convolutional neural networks (CNNs) on edge devices. The framework consists of a pair of edge devices connected via a wireless network: a performance and energy-constrained device D as the first recipient of data, and an energy-unconstrained device N as an accelerator for D. Device D decides on-the-fly how to distribute the workload with the objective of minimizing its energy consumption while accounting for the inherent uncertainty in network delay and the overheads involved in data transfer. These challenges are tackled by adopting the data-driven modeling framework of Markov Decision Processes (MDP), whereby an optimal policy is consulted by D in $O(1)$ time to make layer-by-layer assignment decisions. As a special case, a linear-time dynamic programming algorithm is also presented for finding optimal layer assignment at once, under the assumption that the network delay is constant throughout the execution of the application. The proposed framework is demonstrated on a platform comprised of a Raspberry PI 3 as D and an NVIDIA Jetson TX2 as N. An average improvement of 31% and 23% in energy consumption is achieved compared to the alternatives of executing the CNNs entirely on D and N. Two state-of-the-art methods were also implemented, and compared with the proposed methods.

CCS Concepts: • **Computer systems organization → Embedded systems**; • **Computing methodologies → Computer vision**.

Additional Key Words and Phrases: Internet of Things (IoT), Edge Computing, Energy-Efficient

1 INTRODUCTION

The large scale and rapid pace of deployment of Internet of Things (IoT) makes it extremely cost-sensitive. Consequently, there is an increasing reliance on the *commercial-off-the-shelf* (*COTS*) approach for the design of IoT devices, which are limited in their functionality, storage, performance, and energy capacity [9]. However, because the complexity of the tasks to be performed by IoT devices continues to grow [6, 30, 32, 38], the conventional approach is to perform all the data processing on remote cloud servers. This approach is not scaleable due to the large number of competing devices, long communication delays, the energy overhead of reaching the cloud, and the limitations of the network bandwidth [12, 18].

For example, consider a drone in Figure 1 which needs to run state-of-the-art object recognition algorithms for its navigation [37]. These drones will use low-end COTS processors, and being battery-powered, have limited energy capacity, and consequently, have to rely on cloud servers for most of the processing.

Authors' addresses: Mehdi Ghasemi, Arizona State University, USA, mghasem1@asu.edu; Daler Rakhmatov, University of Victoria, Canada, daler@ece.uvic.ca; Carole-Jean Wu, Arizona State University, USA, caroleje@asu.edu; Sarma Vrudhula, Arizona State University, USA, vrudhula@asu.edu.

1539-9087/2022/4-ART1 $15.00
https://doi.org/10.1145/3530908

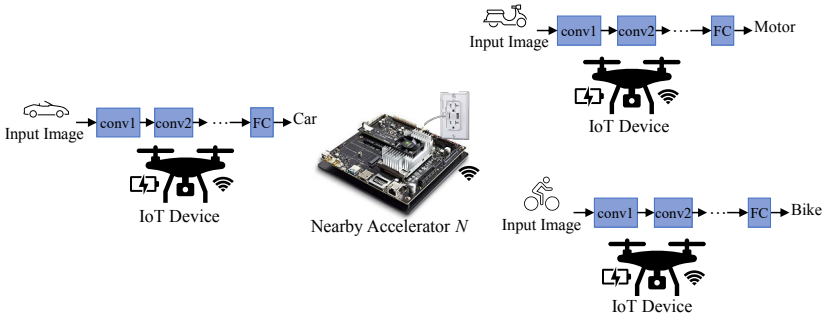


Fig. 1. The convolutional neural networks (CNNs) are needed to be executed on drones with limitation of energy and computational resources.

The aforementioned problem can be addressed by *edge computing* where some or all of the data processing take place closer to the point of data acquisition [29]. The first recipient of the data is the edge device referred to here as the *embedded IoT device D*. Due to the limited capabilities of D, a practical approach is to make available another edge device, i.e., a *nearby wireless network device N*, in close proximity to D, that can share the computational burden of one or more IoT devices. The objective of sharing the computation between D and N is to save the energy consumption of D by having it transfer only the necessary data to N, with N performing the compute-intensive blocks and returning the results to D. This is referred to as *embedded workload reduction* (*EWR*) and its feasibility and utility depends on a number of factors including the granularity of the blocks, the performance and power consumption of the two devices, the communication network bandwidth and delay, and the data transfer scheme between the two devices. With WiFi, this is further complicated by the *uncertainty* in the communication delay, due to possible signal interference, variations in the distance between the devices, etc.

1.1 Overview of Paper

This paper presents a framework called *EdgeWise* which provides an efficient and optimal solution to the *EWR* problem targeting the energy minimization of D assuming that the communication delay between D and N is stochastic. While the general approach presented here can be utilized for various applications, we chose to demonstrate our approach on CNNs not only due to their compute-intensive nature, but also because of their increasing use on edge devices. More advanced networks such as Recurrent Neural Networks (RNNs) [35] used in natural language processing (NLP) and recommender systems [17] require at least an order of magnitude more resources. An input to a CNN, which will be referred to as a *frame*, is processed by a sequence of CNN layers, where each layer is to be executed either on D or on N. A decision as to which device executes a given CNN layer is made by D at the run-time.

Two *EWR* models are described. One, referred to as *fine grained*, involves sampling the network delay prior to the execution of each CNN layer. The other, referred to as *coarse grained*, involves sampling the network delay prior to the execution of an entire frame, and assumes that this value remains unchanged during the execution of all layers. The communication delay is modeled as a sum of two components - a deterministic component that depends only on the amount of data to be transferred, and a stochastic component, referred to as *round-trip time (RTT)*.

For the fine-grained model, a general, non-parametric (i.e. no assumption about the distribution of *RTT*) and data-driven solution is presented, in which the system behavior over time is modeled as a Markov Decision Process (MDP) [7]. A *solution* to the MDP is a *policy*, that specifies an *action* (i.e., selecting D or N to execute

a given CNN layer) in each *state* of the system. The policy is implemented as a lookup table, constructed once offline, and then consulted during the run-time in $O(1)$ time whenever a decision needs to be made. For the coarse-grained model, which is a special case of the more complex fine-grained model, a dynamic programming solution is presented. This optimal solution is constructed by D online in $O(L)$ time, where L is the number of CNN layers. It provides a complete layer assignment in advance, assuming that the measured RTT does not change during the execution of a frame.

In addition to the stochastic nature of the communication delays, another important aspect of the *EWR* problem addressed in this work concerns the *reliability* of the communication channel, which is important in the case of WiFi. If a sequence of CNN layers is assigned to N, and the WiFi connection is suddenly lost, then D would have to recompute all those layers on its own. This situation would make the overall energy consumption of D (as well as the application execution latency) even worse than in the case of N not being used. Hence, the implication of three *data transfer schemes* on the energy consumption of D is explored in this paper: (1) a *conservative* scheme in which the results after executing each layer on N are sent back to D, (2) an *optimistic* scheme in which N sends back results to D only when necessary (e.g., when the next computation is scheduled on D), and (3) a *selective* scheme in which either the conservative or optimistic scheme is selected before a frame is processed, based on a prediction of the network stability.

Extensive evaluation of the proposed solutions are performed on a real platform with a Raspberry PI 3 Model B for D, and an NVIDIA Jetson TX2 for N. The benchmark applications include several well-known compute-intensive CNN algorithms [19, 20, 25]. The computation delay and power consumption of each layer within these algorithms as well as that of the wireless unit were obtained by repeated direct measurements. These were used to determine the fine-grained and the coarse-grained solutions. The experimental results also include the two cases, corresponding to a CNN application being executed entirely on either D (*All-D*), or N (*All-N*). An average improvement of 31% and 23% in energy consumption was achieved compared to these cases. In addition, two state-of-the-art offloading methods ([13, 22]) were implemented and compared with the proposed method in terms of energy efficiency and the overhead involved when making the decisions.

1.2 Summary of Contributions

In summary, the novel contributions of this paper include

- a fine-grained MDP model that accounts for the stochastic communication delays between an energy-constrained embedded IoT device D and a wirelessly accessible nearby accelerator device N, collaboratively executing CNN applications (hosted by D) with the goal of minimizing the energy consumption of D;
- a lightweight use of the optimal fine-grained MDP solution (constructed offline in the form of a policy lookup table), that is consulted online in O(1) time to decide which device, D or N, will execute a current CNN layer, given a currently observed network delay;
- a coarse-grain optimal layer assignment between D and N, constructed online in linear time under the assumption that the network delay measured at the beginning of a given frame stays fixed for the duration of that frame being processed by a CNN, and
- a study of three different schemes for transferring the layer output data from N back to D (when N is responsible for the execution of one or more CNN layers), taking into account the stability of a wireless link between D and N.

1.3 Organization of the Paper

The rest of this paper is organized as follows: The survey of prior work in relation to the *EWR* problem appears in Section 2. Section 3 describes the system components, the communication and energy models, data transfer schemes and a justification of the model assumptions. The fine-grained MDP formulation and its solution method

are presented in Section 4. The coarse grained model, which results in a much simpler, albeit less energy optimal solution, is presented in Section 5. An extensive discussion of the experimental setup and results appears in Section 6 and Section 7. Conclusions are presented in Section 8.

2 PRIOR WORK

The majority of the existing work related to this paper focuses on offloading computation from a high-end mobile device to a cloud server, as opposed to our case that involves a highly constrained low-end IoT device coordinating with a wirelessly accessible accelerator nearby. Different frameworks [10, 14] and methods have been presented in the literature to implement the offloading of computationally intensive code fragments from a mobile device to a cloud server.

Kaya et al. [23] use the call graph of a Java program and bipartition the graph using the FM heuristic method [15]. The method was demonstrated on conventional object recognition algorithms (as opposed to Deep Neural Networks (DNNs)).

Recently, attention has shifted towards DNN algorithms. The computational-intensive nature of these algorithms and their high power consumption makes offloading essential for mobile systems. Jeong et al. [21] present a way to implement DNNs by encapsulating the whole execution state as a web application, which is then transferred to an edge device or cloud server. This approach involves a high overhead of sending the whole model to the edge server.

There are some studies that present computation offloading algorithms where only the data is sent to the cloud server to get the result. *Glimpse* [8] performs real-time object recognition on mobile devices where the entire computation is done on the cloud server. However, when the delay of reaching the cloud server exceeds the frame time, the method uses an active cache of frames to estimate the object classes. This approach relies on the fact that consecutive frames typically do not exhibit sudden scene changes.

MoDNN [31] and *DeepThings* [43] are instances of frameworks that distribute the computation of input data among a set of mobile devices connected using a wireless connection. *Autoscale* [24] is a framework for mid-end mobile devices based on reinforcement learning that selects the execution target for DNNs with the options of CPU, co-processors and an edge device connected in a wireless network. In this framework, the computation of all layers is offloaded to computing resources. *Pipe-it* framework [40] partitions the layers of DNNs across big and LITTLE cores on a mobile MPSoC to achieve higher throughput.

The idea of collaborative execution of DNNs at the granularity of a single layer between the user-end device and the server was demonstrated in frameworks called *Neurosurgeon* [22] and *Edgnet* [28]. The partitions produced by these methods may not be optimal for all cases. In *JointDNN* [13], the overhead of sending the data of different layers is first computed and then used to find an optimal assignment using 0-1 ILP. The above methods perform computation offloading based on the strong assumption that the communication delay is deterministic. In a shared wireless medium, this is most often not the case, and the communication time can vary substantially even over short intervals. Furthermore, the 0-1 ILP optimization approach used in *JointDNN* is generally not scalable and impractical for real-time processing on low-end IoT devices. A comparison of our approach with [22] and [13] is presented in Section 7.3.

Gao et al. [16] consider the variations in the execution times of applications, during code-offloading from a mobile device to a cloud server. The stochastic execution time of an application is modeled using a semi-Markov chain. Estimates of energy savings in the different states of that semi-Markov model are used to decide whether or not a computation should be performed on the cloud server. No optimization is involved in this approach.

The problem of task placement in a distributed system with several users and edge servers, assuming stochastic communication delays was first addressed in [39, 41], and later extended in [5], to maximize a measure of quality of service (QoS) for the users subject to a constraint on the energy consumption of the edge servers. In these

papers, it is assumed that the entire computation takes place on the edge servers (i.e., no collaborative execution between user-end device and edge server) and their focus is on the distributed systems with general workloads.

In this paper, an efficient approach is proposed to address the variability of network delay during the execution of CNNs on a highly resource constrained embedded IoT device that is wirelessly linked to a nearby accelerator device. To minimize its energy consumption, the IoT device executes its CNN applications in tandem with the accelerator device at the granularity of individual CNN layers, while considering the stochastic overhead of communication.

In comparison to the existing literature, the novelty of this work arises mainly from the combination of the following important factors: (1) the primary focus of this work is on low-end embedded IoT devices, as opposed to less-constrained high-end mobile devices coupled with cloud servers, (2) the stochastic communication delay is explicitly taken into account using an effective MDP modeling framework, (3) optimal and lightweight solutions are presented to solve the energy minimization problems at hand, with extensive experimental validation, and (4) the performance of different data transfer schemes is explored for reducing the energy consumption of IoT device.

3 SYSTEM MODEL

In this section, the main components of the system model are identified, and relevant assumptions are described.

Application Model: An application is modeled as a directed acyclic graph (DAG) $G = \{V_G, E_G\}$, where the vertices V_G represent the computational blocks or functions, and the edges E_G represent data dependencies. The nodes of a DAG can be topologically sorted, so that each node is assigned to a *level*. The level of a primary input is 0, and the level of any given node is one plus the maximum level of all its predecessors. A *line graph* is a DAG where each node has a unique level. A CNN can be modeled as a line graph [42], in which each layer (node) constitutes a single block of computation.

Processing Elements: As stated in Section 1, there are two main processing engines: an embedded IoT device D with limited energy capacity and limited performance, and a nearby accelerator device N that can deliver much higher performance and is not energy-constrained. It is assumed that the application code is available on both D and N, and only the data captured by D and block-specific parameters are transmitted to N as needed. When D decides to let N execute some computational block, D uses its WiFi port to transfer the input data of that block to N and then to receive the results from N. While N is performing its assigned computations, D listens to its WiFi port waiting to receive the data from N.

3.1 Communication Delay Model

The communication delay between D and N using a wireless link exhibits substantial variations. This delay is affected by a number of factors, such as the communication distance, packet size, the traffic load, interference by neighboring devices, etc. It is commonly expressed as ([4])

$$\delta_{D,N} = S/B + RTT, \tag{1}$$

where RTT is the *round-trip time* or network delay (ms), B is the bandwidth (Mbps), and S is the size of the data (Kb). The bandwidth of the wireless communication channel (B) is assumed to be fixed. On the other hand, RTT captures all the sources of network delay variations, and consequently, is assumed to be a non-negative random variable with an arbitrary distribution. Having both bandwidth and RTT as random variables, the communication delay can be modeled as a sum of two random variables.

Figure 2 illustrates the significance of the wireless RTT, observed in our experimental setup consisting of a Raspberry PI 3 Model B (RPI), serving as D, and an NVIDIA Jetson TX2 (Jetson), serving as N. Note the substantial variations in communication delay when using WiFi as compared to wired LAN communication. Although

the round trip time varies for a wired connection as well, the magnitude of those variations is negligible in comparison with the execution time of computationally heavy nodes in the CNN flow graph.

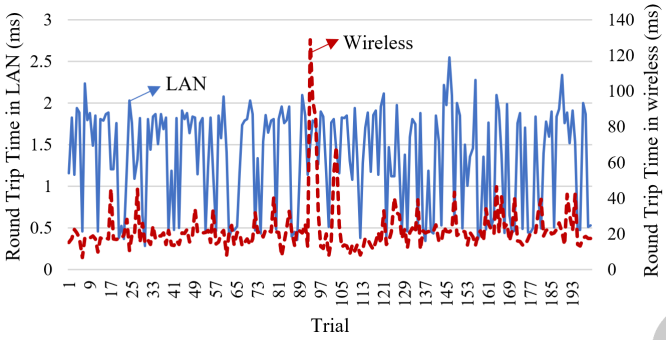


Fig. 2. Round-trip time between IoT device D and accelerator device N when using wired LAN (left axis) and WiFi (right axis) links. The magnitude and stochastic behavior of the wireless *RTT* is a significant factor during collaborative application execution involving wireless data transfers between D and N. Note the difference in the scale of coordinates.

3.2 Energy Model

Our objective is to determine the best assignment of computation blocks (represented by nodes in V_G) to the devices D and N such that the energy consumption of D is minimized during the application execution. Given $G = (V_G, E_G)$, let $m_i = 0$ or 1 if node $i \in V_G$ is to be executed on D or N, respectively. The energy consumption of D, denoted by E_D, consists of several components, described below. Section 6 discusses how realistic estimates of these quantities are obtained.

(1) $E_D(i|D)$ and $E_D(i|N)$ denote the computation energy consumption of D when the computation block represented by node $i \in V_G$ is executed on device D and device N which is dependent on the execution time on D and N ($\delta(i|D)$ and $\delta(i|N)$), respectively. Thus, $E_D(i|N)$ is equal to the energy consumption of D when it is idle and listening to its WiFi port for incoming data.

(2) $E_D(i|x, j|y)$ denotes the communication energy consumption due to the data transfer associated with edge $(i, j) \in E_G$, when node i is executed on device x and node j is executed on device y, where $x, y \in \{D, N\}$. The meaning of each of the four possibilities is explained below.
   (a) $E_D(i|D, j|N)$ is the communication energy consumption of D when node i is executed on D and node j is executed on N. As $(i, j) \in E_G$, the output of node i is the input to node j. Therefore, $E_D(i|D, j|N)$ is the energy consumed by D when transferring the results of node i from D to N.
   (b) $E_D(i|N, j|D)$ is the communication energy consumption of D when node i is executed on N and node j is executed on D. As $(i, j) \in E_G$, the output of i is the input to j. Therefore, $E_D(i|N, j|D)$ is the energy consumed by D when receiving i's results from N.
   (c) $E_D(i|N, j|N)$ has two interpretations. In the case of our conservative data transfer scheme (where the results of i computed by N must be transferred back to D), $E_D(i|N, j|N)$ is the same as $E_D(i|N, j|D)$. On the other hand, in the case of our optimistic data transfer scheme (where D does not receive a copy of i's results), $E_D(i|N, j|N)$ is zero.
   (d) $E_D(i|D, j|D) = 0$. As nodes i and j are both executed on D, there is no communication energy consumed by the data transfer.

The computation and communication energy consumption of D can now be expressed in terms of the above quantities and the decision variables as follows:

$$E_D(i) = m_i E_D(i|N) + (1 - m_i) E_D(i|D), \tag{2}$$

$$E_D(i,j) = m_i m_j E_D(i|N, j|N) + m_i(1 - m_j) E_D(i|N, j|D) + (1 - m_i) m_j E_D(i|D, j|N). \tag{3}$$

Our objective is to minimize the total energy consumption of D during the application execution, including the computation energy and communication energy. This is denoted by E_{tot}.

$$Minimize \quad (E_{tot} = \sum_{i \in V_G} E_D(i) + \sum_{(i,j) \in E_G} E_D(i,j)) \tag{4}$$

3.3 Data Transfer Schemes

The IoT device D, which is the first recipient of the data, serves as the *master*, on which the layer assignment decisions are made. If a computation block $i \in V_G$ is delegated to N, then D transfers all the necessary data to N to initiate the execution of i. Besides deciding on which device (D or N) a given block should be executed, another important factor to be considered is *when* the results of computations on N should be transferred back to D. This is referred to as the *data transfer scheme*. In the presence of an *unstable* communication medium, this can have a substantial impact on the energy consumption of D because when its WiFi port becomes unresponsive, the currently delegated computations (performed by N) must be entirely redone by D.

To simplify the description of the data transfer schemes in question, consider the execution sequence $(D_1, N_2, N_3, N_4, D_5, D_6, N_7, N_8, D$ D_i (N_i) indicating that the computation block i is executed on D (N), and let o_i denote the data outputs generated by i.

- **Conservative Scheme**: This requires that the results of every computation block executed on N be transmitted immediately back to D. For the above sequence, the set of data outputs transmitted back to D would be $(o_2, o_3, o_4, o_7, o_8)$. This ensures the robust execution of the application even when N suddenly becomes unavailable for whatever reason, in which case D can use the previous results obtained from N without having to redo all of N's computations. For example, if D fails to receive o_4 in time, it can take over starting from computation block 4, without having to recompute blocks 2 and 3.
- **Optimistic Scheme**: This allows for the results to be sent from N to D only when necessary. That is, N would transmit the results of a block it is executing only if the next block is to be executed by D. For our example sequence, only the results (o_4, o_8) would be transmitted to D. From an energy perspective, this is riskier because if D fails to receive o_4 in time, D would have to recompute blocks 2 and 3.
- **Selective Scheme**: Unlike the conservative and optimistic schemes, this scheme aims to take into account the future state of the communication medium. Figure 3 depicts how the selective scheme works. Based on the past RTT measurements, it predicts whether or not the network will be *unstable* prior to processing the current frame. Based on the outcome of that prediction, it selects either the conservative or optimistic scheme, and then determines the corresponding optimal assignment of computation blocks to be executed on D or N.

For the selective scheme, a simple, lightweight criterion is used to define the instability of the WiFi link, whose behavior is modeled using a Markov chain by discretizing the distribution of the RTT values into r intervals $(I_1, I_2, \cdots, I_r)$. The WiFi link is considered to be in *state* k if $RTT \in I_k$, for $1 \leq k \leq r$. Given a timeout value θ, the WiFi link is deemed unstable if $RTT > \theta$. It is assumed that $RTT_{min} \leq \theta \leq RTT_{max}$. Let $\kappa(\theta)$ be the index such that $\theta \in I_\kappa$, let $p_{k,\ell}$ denote the one-step k-to-ℓ state transition probability of the WiFi link, and let q_k denote the total probability that the next state of the WiFi link will be unstable given that its present state is k. That is,

$$p_{k,\ell} = \text{Prob}(\text{Next Frame } RTT \in I_\ell \mid \text{Current Frame } RTT \in I_k) \tag{5}$$

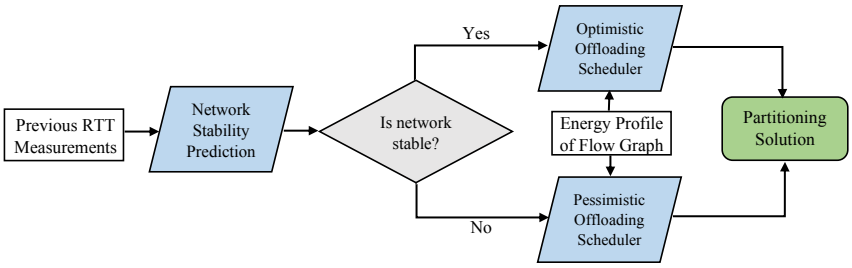


Fig. 3. The overall view of two-level controller. The network stability prediction unit determines the data transfer scheme and the scheduler assigns the mapping of graph nodes to devices.

$$q_k = \sum_{\ell=\kappa(\theta)}^{r} p_{k,\ell}. \tag{6}$$

Now the selective scheme depicted in Figure 3 chooses the conservative scheme if $q_k \geq 0.5$, otherwise it selects the optimistic scheme.

3.4 Discussion of Model Assumptions

Application Model: Restricting the graph model of computation to a line graph is well-suited for CNNs and, as will be demonstrated in the experimental results, is a valuable special case. The presented method can be applied for other workloads in the form of a line graph with different performance profiles. The proposed solution methods can be extended to general DAGs currently under development.

Network Delay: Our wireless communication delay model expressed by Equation 1 is a well established lumped model [4] that can account for various sources of variability such as packet drop, packet processing delays and others, captured by the random RTT term. The empirical distribution of the RTT is characterized during the profiling step, and used in constructing our fine-grained solution. It was also observed that the variation in the execution time on N was not significant compared to the variation in RTT.

Objective Function: The energy consumption of D is the product of the total delay (including computation and communication delays) and the power consumption of D. Thus delay is also accounted for when considering energy as the objective function. Furthermore, with line graphs, E_D can be used to represent delays (instead of energy consumption), which will translate our original problem into delay minimization *without any changes* to the modeling approach or algorithms. Also, note that our energy objective function focuses exclusively on D, assuming that the limited capacity of D's energy source is of utmost concern. If accounting for the energy consumption of N (powered by its own source) is also important, it can be incorporated by adding related quantities to $E_D(i|N)$, $E_D(i|N, j|D)$, and $E_D(i|D, j|N)$.

Choice of Experimental Platform: Two popular commercial off-the-shelf (COTS) devices (RPI and Jetson) in IoT systems have been selected serving as D and N. However, the methods described here are applicable to other platforms once the profiling step is done. In the scenario where N is serving several other IoT devices, its utilization affects the execution time of layers on N as well. Thus, this changes the energy cost of executing a layer on N. The presented solution here can account for such a variation using the term $E_D(i|N)$ in the formulation.

Offloading benefit: The benefit of offloading in terms of energy consumption is dependent on the overhead of communication energy and the computation energy profiles on devices. The stochastic variable that significantly affects the communication energy is the communication delay which changes the energy-optimal partitioning. Our proposed energy optimization method is orthogonal to any other existing method for saving the power

consumption of D. Given the power profiles of the CNN layers, our approach provides the energy-optimal partitioning across devices. In the evaluation of our approach, the power consumption values for the layers of a single CNN are the same throughout different comparisons on the same CNN (e.g. comparison with All-N and All-D).

4 FINE-GRAINED SOLUTION

This section describes a solution to our workload reduction problem accounting for the stochastic variations in the communication delays during the execution of a CNN application. The problem at hand can be viewed as a sequential decision process whereby D decides on which device, D or N, a block will be executed. Our approach is to model the time evolution of the application execution as a *Markov Decision Process* (MDP) [36]. The proposed solution has three valuable characteristics: (1) it has $O(1)$ complexity; (2) it is data-driven with minimal assumptions about the data, i.e., it does not depend on complex parametric models; and (3) it produces an optimal assignment of computation blocks.

The outcome of an MDP is a *policy*, which specifies the *action* to take (i.e., assign the block to D or N) for each possible *state* of the system. Therefore the policy is simply a table representing a function from *states* to *actions*. Once such a policy is determined, the actual step that D performs is very simple: before the execution of a given block (respecting the partial order specified by the application dataflow graph G), D determines the *present state* of the system, and based on that state, consults the policy in $O(1)$ time (table lookup), and determines the action.

Our MDP model requires a restriction: the application dataflow graph G is topologically sorted, and all nodes with the same layer are combined to form a single block. Thus G is transformed into a *line graph* representing a sequence of blocks to be executed, where all the outputs of any given block are relayed to the next block in the sequence as inputs. With this restriction, our MDP model assigns individual blocks to either D or N. Note that in the context of CNNs, the terms block and layer become synonymous.

4.1 Markov Decision Processes

There is an extensive body of literature on MDPs, as it is a rich and well-developed subject with many variations [36]. The essential components, and one of several methods of constructing an optimal policy (namely value iteration [7]) are described next.

An MDP is represented as a tuple $(S, A, P, COST, \gamma)$ where:

- A is a finite set of actions. In our case, there are only two actions: assigning a block to either D (local execution) or N (remote execution).
- S is a finite set of states. A state $s \in S$ is a tuple (Δ, i, d), where i denotes the current block or CNN layer, and d denotes the device on which i is executed and Δ is the denotes the interval within which the present RTT value belongs. In our model of the communication delay, the round-trip time (RTT) is a random variable. The range of RTT is partitioned into r equilength intervals $I_1, I_2, \cdots, I_r$. All that is being done is approximating RTT to some value in a finite discrete set. Making a discrete approximation allows us to enumerate the state space when constructing the MDP solution. Likewise, the range of the bandwidth variation can be divided into intervals and added to the state definition. The value of Δ is obtained by measuring the network delay just prior to taking an action. The total number of states in the MDP is $n = 2rL + r$, where r and L denote the number of RTT delay intervals and the number of layers respectively. The value of n determines the number of entries in the lookup table that is used to select the corresponding action for the current state. The total number of states includes r dummy states which represent the initial delay. The dummy states only include Δ in their definition.
- $P(s_1, a, s_2) = Pr(s_{t+1} = s_2 | s_t = s_1, a_t = a)$ is a state transition probability function, i.e., the probability of transitioning from a present state s_1 at decision epoch t to the next state s_2 at decision epoch $t + 1$, with

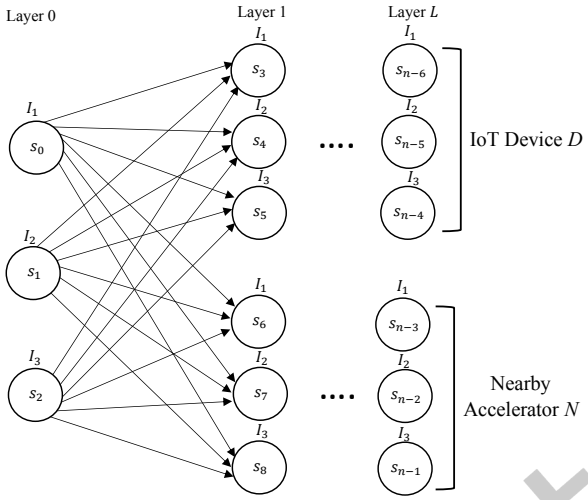


Fig. 4. MDP modeling example for three intervals of network delay.

action a. These state change probabilities are the probabilities associated with entering a specific network delay interval from a current interval. Details on how these probabilities are obtained (using network delay profiling) are discussed in Section 6 describing our experimental setup.

- $COST(s_1, a, s_2)$ is the cost incurred after transition from state s_1 to state s_2 due to action a. The cost values are the sum of computation and communication energy of D for the selected action.
- $\gamma \in [0,1]$ is the discount factor that accounts for the difference between present and future costs.

The purpose of using discount factor γ in the MDP formulation is to compare the policies in the infinite time horizon and evaluate the convergence of the MDP policy. Infinite time horizon refers to the case where the stochastic system is considered from decision epoch 0 to ∞. Optimization over the infinite time horizon essentially ensures the solution optimality regardless of when the process ends. Since the total incurred cost of performing actions in an infinite number of decision epochs would be infinity, there is a need to make the total cost finite when comparing policies. Thus, the cost at each decision epoch is multiplied by the discount factor γ to make the distant future costs close to zero.

Figure 4 shows a graphical representation of an MDP for a set of computation blocks (layers) connected in the form of a chain, in which the network delay falls into one of three discrete intervals. As a result, there are three initial states. In each state, an action (local execution or remote execution) is selected. For each layer from 1 to L (number of layers), the top three states correspond to the action 0 (i.e., local execution on D) and the lower three states correspond to action 1 (i.e., remote execution on N).

4.2 Constructing an Optimal Policy

In the MDP framework, after selecting an action a in each state s, the system transitions to another state with a probability P and the action will lead to a cost value $COST$. The goal is to find the best policy π which assigns an action to every state to minimize the following cost function:

$$\sum_{t=0}^{\infty} \gamma^t COST(s_t, a_t, s_{t+1}), \tag{7}$$

where a_t denotes the action in state s_t in the obtained optimal policy. The MDP solution should consider the probabilities of state change and the cost values obtained in different decision epochs after selecting a specific action. In fact, a greedy action in each state does not necessarily lead to the optimal solution in the long run, and the MDP should take into consideration all the state changes and the cost values obtained in the subsequent decision epochs. Hence dynamic programming is used to implicitly consider all the possibilities.

In order to find the MDP solution, which is a policy that assigns the optimal action to each state, one can use either policy iteration or value iteration [7]. Our MDP problem is solved using the latter, as shown in Algorithm 1, in which $V(s)$ denotes the cost value in state s. For all states, the algorithm calculates the optimal value of $V(s)$ iteratively, considering all possible actions and next states $s^{'}$ until the difference ϵ in the value of $V(s)$ in subsequent iterations becomes negligible (less than a small number ρ which determines the stopping condition and therefore convergence). In our case, the actions are either 0 (execute on D) or 1 (execute on N), and the associated cost values are stored in $D[s]$ and $N[s]$. The cost values account for the probability of state change and the discount factor γ. Finally, the method finds the optimal policy (action) in each state s based on *arg min* operator.

Algorithm 1 Fine-Grained Value Iteration (FGVI)

$V(s) = 0, \quad \forall s \in S$ (set of states)
repeat
  $\epsilon = 0$
  for each $s \in S$ **do**
    $v = V(s)$
    $D[s].COST = \sum_{s'} P(s, 0, s^{'}) \times (COST(s, 0, s^{'}) + \gamma V(s^{'}))$
    $N[s].COST = \sum_{s'} P(s, 1, s^{'}) \times (COST(s, 1, s^{'}) + \gamma V(s^{'}))$
    $V(s) = min(D[s].COST, N[s].COST)$
    $\epsilon = max(\epsilon, |v - V(s)|)$
  end for
until $\epsilon < \rho$
$\pi(s) = argmin_a \sum_{s'} P(s, a, s^{'}) \times (COST(s, a, s^{'}) + \gamma V(s^{'}))$

To summarize, our fine-grained MDP approach involves the following steps:

- Profiling: The behavior of the system is profiled using a sufficient number of repeated experiments. The probability of state change and the cost function for different actions are obtained. These values are then formatted as an MDP grammar file.
- Solving the MDP problem: After constructing the MDP problem including the probabilities and cost function, the MDP is solved and the optimal policy for selecting the suitable action is stored in the lookup table.
- Using the lookup table: During the run-time, the lookup table is consulted to select the suitable action depending on the current state. An MDP evaluator tests next states to find the incurred cost values on average.

Complexity of Decision: Once the optimal policy is determined offline using the cost values obtained by profiling, the optimal action in each state is simply determined by a table lookup, which is of $O(1)$ time complexity.
Reduction to Coarse-Grained Solution: In special case of a fixed network delay (i.e., when the RTT does not vary across multiple delay intervals), the optimal MDP solution would reduce to a fixed assignment of all individual layers, which can be computed at the beginning of the frame in advance (i.e., before starting the CNN application execution). In fact, the probability of state change would be one for all the state transitions ($P(s, a, s^{'}) = 1$), with $\gamma = 1$. Additionally, the stopping condition ($\delta < \rho$) would still hold in the fixed delay case. Thus, the recursive

relationship between $V(s)$ and $V(s')$ (which are the respective cost values in the current state s and the next state s') given by $V(s) = min(\sum P(s,0,s^{'})(COST(s,0,s^{'}) + \gamma V(s^{'})), \sum P(s,1,s^{'})(COST(s,1,s^{'}) + \gamma V(s^{'})))$ would reduce to $V(s) = min(\sum COST(s,0,s') + V(s'), \sum COST(s,1,s') + V(s'))$. In other words, the value iteration procedure becomes much simpler, and the following key points are to be noted:

- In the case of a fixed RTT, it is no longer necessary to perform the offline profiling of the MDP state transition probabilities. Instead, the network delay measured at the beginning of the frame is assumed to be fixed for the duration of that frame. Such a coarse-grained model still allows for the RTT changes from frame to frame (but not within a frame from layer to layer, which is handled by our fine-grained MDP model).
- In the coarse-grained case, Algorithm 1 (executed offline to generate the lookup policy table in the fine-grained case) can be replaced by a low-complexity algorithm executed at the beginning of a frame at the run-time, given the measured RTT whose value affects the communication energy costs. The details of this algorithm are presented in the next section.

5 COARSE-GRAINED SOLUTION

The assumption that the RTT remains constant during the processing of a frame leads to the following simplification of the MDP formulation.

Under the assumption of a fixed RTT, wireless transmissions and receptions are modeled as graph nodes shown in Figure 5. The costs of computation and communication are associated with the edges of the graph. Before executing a block or layer, the input data of that block needs to be received. After receiving the data, there is a cost associated with running the block on D or N which is depicted in the model as $E_D(i|D)$ or $E_D(i|N)$ (shown inside the box of each layer). The communication costs between different layers can be one of the four cases shown in the graph. For instance, one case of the previously mentioned possible cost values is $E_D(i|D, i+1|N)$ when the layer i is executed on D and layer $i+1$ on N.

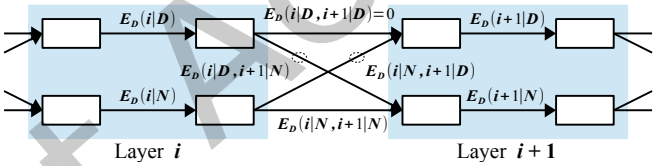


Fig. 5. Simplified graph model in the special case of a fixed network delay during application execution.

Two special nodes, labeled S (source) and T (sink) are also introduced, such that S transmits the input data to be processed in the first layer, while T receives the result of the last layer. The cost of S- and T-connected edges is set to zero; however, $E_D(1|N)$ and $E_D(L|N)$ are adjusted accordingly, to account for the initial D-to-N and final N-to-D communication costs.

Given the graphical model depicted in Figure 5, the EWR problem with the deterministic communication delay can be solved using Dijkstra's algorithm. The time complexity of finding a shortest path solution would be $O(|V_G|^2)$ when a graph is represented by its adjacency matrix. However, the special structure of the underlying graph allows for a much simpler method described below.

First, block arrays D[i] and N[i] are introduced representing computation within a given block i. Each D[i] has a Cost attribute, equal to the sum of $E_D(i|D)$ and the minimum accumulated cost of reaching the block i from S (source) when the block is scheduled to be executed on D. Similarly, each N[i] has a Cost attribute, equal to the sum of $E_D(i|N)$ and the minimum accumulated cost of reaching the block i when the block is scheduled to be executed on N.

To update D[$i+1$].Cost, only three quantities are required: $E_D(i+1|D)$, $E_D(i|D, i+1|D)$ summed with D[i].Cost, and $E_D(i|N, i+1|D)$ summed with N[i].Cost. Then,

$$\mathrm{D}[i+1].\textsc{Cost} = E_D(i+1|D) + \min(\mathrm{D}[i].\textsc{Cost} + E_D(i|D, i+1|D), \mathrm{N}[i].\textsc{Cost} + E_D(i|N, i+1|D)). \quad (8)$$

Similarly,

$$\mathrm{N}[i+1].\textsc{Cost} = E_D(i+1|N) + \min(\mathrm{D}[i].\textsc{Cost} + E_D(i|D, i+1|N), \mathrm{N}[i].\textsc{Cost} + E_D(i|N, i+1|N)). \quad (9)$$

Algorithm 2 shows the resulting partitioning method. After initialization steps 1-5, it performs $L-1$ iterations: steps 7-9 implement Equation (8), while steps 10-12 implement Equation (9). Upon completion of this loop, step 14 identifies the end point ($i = L$) of the minimum-cost path, and step 16 recovers the optimal mapping sequence by tracing the Pred pointers (recorded predecessors) back to the starting layer ($i = 1$). During this traceback, the 0-1 Type indicators yield the sought bit-values of the assignment vector m[i]. The space/time complexity of our algorithm is $\Theta(L)$.

Algorithm 2 Coarse-Grained Dynamic Programming (CGDP)

1: **for** each $i = 1, 2, ..., L$ **do**
2: $D[i].TYPE = 0, D[i].COST = \infty, D[i].PRED = null$
3: $N[i].TYPE = 1, N[i].COST = \infty, N[i].PRED = null$
4: **end for**
5: $D[1].COST = E_D(1|D)$, $N[1].COST = E_D(1|N)$
6: **for** each $i = 1, 2, ..., L-1$ **do**
7: $c1 = D[i].COST + E_D(i|D, i+1|D) + E_D(i+1|D)$,
8: $c2 = N[i].COST + E_D(i|N, i+1|D) + E_D(i+1|D)$
9: **if** $c1 \le c2$ **then** $D[i+1].COST = c1, D[i+1].PRED = D[i]$
else $D[i+1].COST = c2, D[i+1].PRED = N[i]$
10: $c1 = D[i].COST + E_D(i|D, i+1|N) + E_D(i+1|N)$,
11: $c2 = N[i].COST + E_D(i|N, i+1|N) + E_D(i+1|N)$
12: **if** $c1 \le c2$ **then** $N[i+1].COST = c1, N[i+1].PRED = D[i]$
else $N[i+1].COST = c2, N[i+1].PRED = N[i]$
13: **end for**
14: **if** $D[L].COST \le N[L].COST$ **then** $v = D[L]$ **else** $v = N[L]$
15: **for** each $i = L, L-1, ..., 1$ **do**
16: $m[i] = v.TYPE, v = v.PRED$
17: **end for**

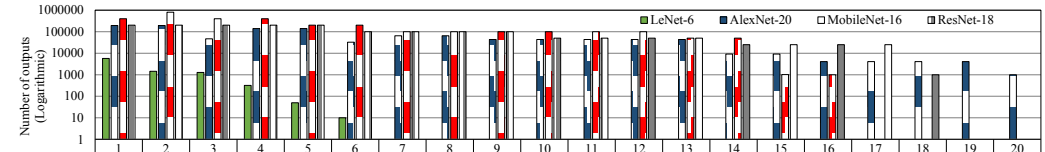
6 SETUP OF EXPERIMENTS

This section discusses the CNN applications and experimental devices in use, as well as their computation and communication profiling details.

6.1 CNN Benchmarks

Four well-known CNNs were used as test cases: LeNet [27], AlexNet [25], MobileNet [20], and ResNet18 [19], having 6, 20, 16, and 18 layers respectively. LeNet was tested on the MNIST dataset and the others on the ImageNet dataset using images of size $224 \times 224 \times 3$ [11]. The layers in the CNNs are treated as the computational blocks to be offloaded as they correspond to the most time consuming code portions.

The amount of data to be transferred between D and N, in conjunction with the stochastic RTT, determines the energy costs associated with communication and affects the optimal assignment solution. Figure 6 shows the



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LeNet-6	5760	1440	1280	320	50	10														
AlexNet-20	193600	193600	46656	139968	139968	32448	64896	64896	43264	43264	43264	43264	43264	9216	9216	4096	4096	4096	4096	1000
MobileNet-16	401408	802816	401408	401408	200704	200704	100352	100352	100352	100352	100352	100352	50176	50176	1024	1000				
ResNet-18	200704	200704	200704	200704	200704	100352	100352	100352	100352	50176	50176	50176	50176	25088	25088	25088	25088	1000		

Fig. 6. Number of output data bytes for each layer of different CNNs. The amount of output data decreases substantially towards the end layers of the CNNs.

distribution of the number of output data bytes per layer for the four CNNs. The decrease in the amount of data in the final layers provides an incentive to offload those layers to the nearby accelerator. Note that LeNet is the least expensive in terms of layer-to-layer data movement, while MobileNet is the most expensive.

The following example illustrates the impact of uncertainty in RTT, given in equation (1). Suppose layers 1 and 2 of MobileNet are executed on D and N, respectively, thus requiring a D-to-N data transfer of size $S = (401408 \times 8)/1024 = 3136$ Kb. Assuming the bandwidth $B = 12$ Mbps, the communication delay of $\delta = 255$ ms $+ RTT$ is obtained. The range of RTT values from 7 ms to 130 ms (see Figure 2 for example) will result in the variation in δ between 3% and 51%. Similarly, if layers 10 and 11 of ResNet18 are executed on D and N, respectively, then the corresponding communication delay may range from $32(S/B) + 7(RTT)$ ms to $32(S/B) + 130(RTT)$ ms. Clearly, the stochastic RTT contribution must be taken into account, which is the key motivation behind this work.

6.2 Computation and Communication Profiling

To demonstrate the value of the proposed approach, the Raspberry Pi 3 Model B (RPI) [2] was selected as the IoT device D and the NVIDIA Jetson TX2 (Jetson) [1] as the nearby accelerator N. The devices were connected using a UniFi AP AC LITE wireless access point [3]. The following sections contain a description of how computation (performance and power consumption) and communication (network delay) profiling was done.

6.2.1 Performance Profiling. The performance of each layer in different CNNs was individually profiled on both D and N. This involved executing each layer several times and measuring the execution time using built-in time functions in Python. Figure 7 shows the ratio of the execution times on D to those on N for the different CNNs.

The convolution layers were the most time consuming on both devices. For LeNet, the Python *numpy* package was used; this package did not utilize the GPU available on N. On the other hand, *PyTorch* [34] was used for profiling AlexNet, MobileNet, and ResNet18, which utilized the GPU on N. This explains the differences in relative speedup of these CNN layers on N versus D. For instance, for LeNet, a speedup of 3X was obtained using the Jetson (N) over the RPI (D), whereas the speedup for the others ranged from two to five orders of magnitude due to the use of GPUs. Note that for fully connected layers (layers 15 of AlexNet, 16 of MobileNet, and 18 of ResNet18), the RPI was faster than the Jetson due to the lower overhead of loading the weights into main memory. Another important observation is that the execution times of MobileNet layers are much higher than for the other CNNs. Consequently, the network delay becomes less significant when offloading MobileNet layers in comparison to the other CNNs.

6.2.2 Energy Profiling. The energy consumption of D (i.e. multiplication of power consumption and delay) including computation and communication energy was measured during the profiling step. The average power

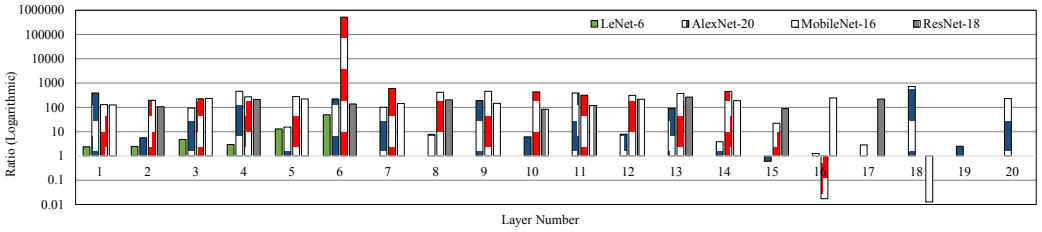


Fig. 7. The ratio of execution time on D to execution time on N ($\frac{\delta(i|D)}{\delta(i|N)}$) for different layers of CNNs. Some layers are executed slower on N due to higher load time of weights (fully connected layers).

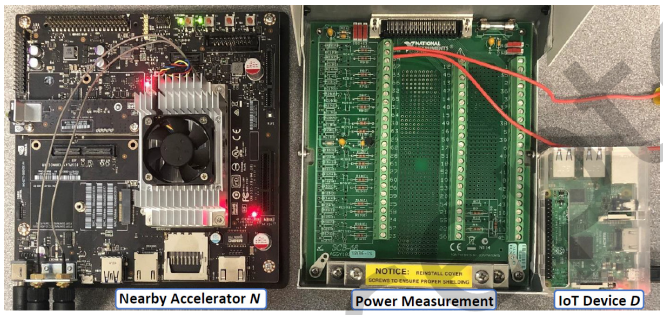


Fig. 8. Experimental setup.

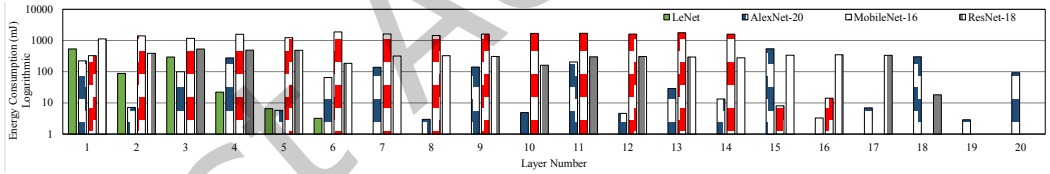


Fig. 9. Energy consumption of executing different layers of CNN models on IoT device D.

consumption of running CNNs on D was measured for individual layers of each CNN benchmark. The measurement was done using a National Instrument PXI digital acquisition unit (Figure 8). The total current drawn using the RPI was measured placing a 0.1Ω resistance along the path. The current obtained was multiplied by the supply voltage to obtain the power consumption. The power consumption of running ResNet and MobileNet was higher than AlexNet and LeNet. This was verified by checking the CPU utilization of RPI while running the CNNs. Figure 10 shows the CPU utilization of RPI which shows higher utilization for MobileNet and ResNet compared with AlexNet and LeNet. Figure 9 shows the energy consumption of all the layers in the CNNs. The highest total energy consumption was related to MobileNet. The communication energy is the product of the communication power and the data transfer time. Communication energy profile was used in the decision making of layer partitioning.

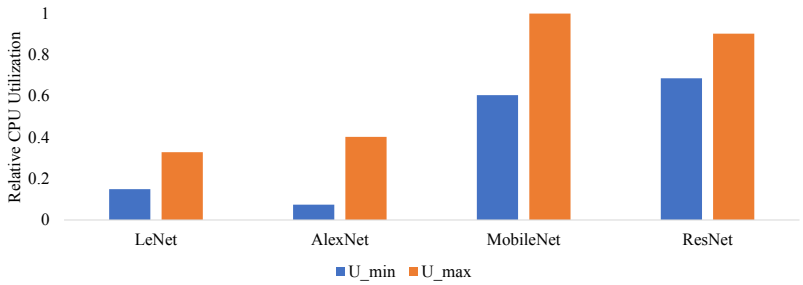


Fig. 10. The CPU utilization of PRI while running different CNNs. The utilization numbers are normalized to the maximum utilization.

6.2.3 Network Delay Profiling. For the MDP solution, the state transition probabilities associated with the communication network were estimated by repeated measurements of the RTT, and assigning the values to the appropriate network delay interval. This yields the discrete probability distribution of the network delay. Estimates of state transition probabilities were obtained by counting the frequency with which interval I_j was observed following interval I_i, over a sequence of measurements.

7 EXPERIMENTAL RESULTS

In this section, the computation offloading (i.e., IoT device workload reduction) results are presented which are obtained using: (1) our fine-grained MDP approach in the stochastic network delay case, and (2) our coarse-grained dynamic programming procedure in the fixed network delay case. It should be noted that in both cases, the corresponding assignments of individual CNN layers to either D or N are *optimal solutions* to the energy minimization problem stated in Section 3.

7.1 Fine-Grained Solution

In order to build the MDP model, the system must be profiled first. This includes taking repeated direct measurements of the network delay and constructing the state transition probabilities mentioned in the MDP formulation (Section 4). The cost functions were also calculated based on the profiling of both devices. The states, probabilities, and cost functions were then converted into the MDP grammar format. To evaluate the convergence of the method to the optimal solution in an infinite time horizon, discount factor γ for the MDP was set to 0.9. The public domain framework called APPL [26, 33] was used to solve the MDP problem with the given cost functions and state change probabilities. Then, the policy was computed and expressed in the form of a lookup table specifying the suitable action in each state. The number of entries in the lookup table (as shown in Table 1) was at most 287 for AlexNet when the network delay distribution was partitioned into seven network delay intervals. In other words, the overhead of looking up the state and action pair in the lookup table is negligible.

The MDP lookup table generation is only done once if the communication model is fixed. There can be several generated lookup tables for different communication models that can be used for different scenarios. The overhead of generating the MDP lookup table for $r = 7$ on a Core-i7 3612 CPU with 8GB RAM is reported in Table 2. The measured overheads demonstrate that the MDP can be re-trained at run-time where the model is deviating from the actual model.

Table 3 shows the results of applying the MDP to different CNNs under the conservative data transfer scheme (see Section 3). These numbers were obtained after running the experiments 200K times. It shows that using more network delay intervals for the MDP formulation leads to the lower energy consumption of D. This clearly

Table 1. The number of entries in the MDP lookup table (i.e., number of states). The overhead of looking up the table is negligible.

	Number of network delay intervals (r)		
CNN	$r = 3$	$r = 5$	$r = 7$
LeNet	39	65	91
AlexNet	123	205	287
MobileNet	99	165	231
ResNet18	111	185	259

Table 2. The overhead of computing the MDP lookup table for $r = 7$.

CNN	Overhead (ms)
AlexNet	713
LeNet	40
MobileNet	258
ResNet	343

Table 3. Fine-grained solution: Average energy consumption of D for 200K experiments using different number of network delay intervals (conservative scheme).

	Number of network delay intervals (r)			
CNN	$r = 1$	$r = 3$	$r = 5$	$r = 7$
LeNet	0.708 J	0.468 J	0.432 J	0.416 J
MobileNet	8.68 J	6.876 J	6.628 J	6.515 J
AlexNet	2.077 J	1.855 J	1.798 J	1.778 J
ResNet18	6.587 J	5.574 J	5.247 J	5.099 J

demonstrates the importance of accounting for the variations in the communication delays. The column labeled $r = 1$ corresponds to using the midpoint of the entire network delay range ($(RTT_{min} + RTT_{max})/2$), which is the coarse-grained special case of the fixed $RTT = 67.925\ ms$. Using more than seven intervals did not result in any further decrease in the energy consumption. Comparing the results of $r = 1$ with $r = 7$, the energy consumption of D was reduced approximately by 41% for LeNet, 25% for MobileNet, 14% for AlexNet and 22% for ResNet. The average reduction in energy consumption was 25.5%. In Section 7.2, additional coarse-grained special cases of different fixed values of the RTT equal to 10, 20, 40, 80, and 120 ms are discussed.

Using $r = 7$ delay intervals resulted in the most energy savings for D. For this case, some CNN layers were always assigned to N (Jetson), while some other layers were never assigned to N (see Figure 11). This shows the potential for model order reduction in the context of our MDP framework, which may in turn give rise to efficient online algorithms incorporating both fine-grained (stochastic) and coarse-grained (deterministic) optimization approaches.

Table 4 shows the results of simulating all three data transfer schemes repeatedly for 1000 trials. The number of the network delay intervals was $r = 7$. For the selective scheme, the value of the timeout was $\theta = 60ms$ and the simulations of the selective scheme included the presence of failures, whereas the other two were assumed to be ideal. The value of θ is different than $RTT = 67.925ms$ which was used for comparing the MDP solution in the case of $r = 1$. The results clearly show that even when failures are present, the selective scheme results in much less energy consumption by D when compared to the conservative scheme, and nearly the same as the (ideal)

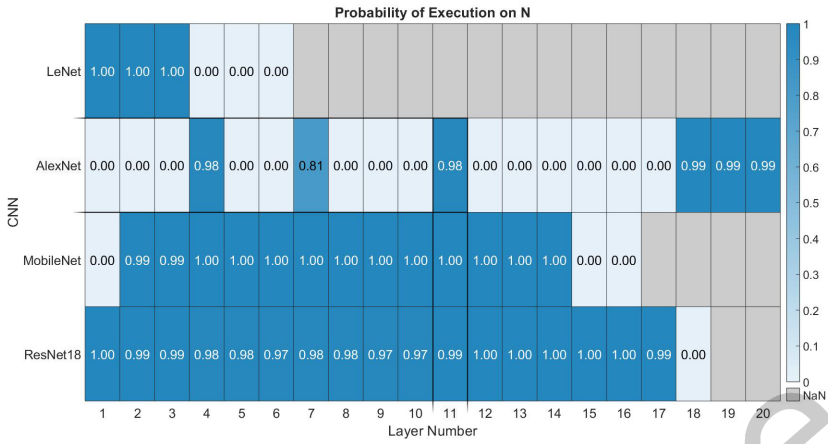


Fig. 11. Fine-grained approach: The probability of executing a layer on N for different CNNs ($r = 7$). Some layers were never executed on N.

Table 4. Fine-grained solution: Mean (μ) and standard deviation (σ) of energy consumption of D (in Joules) for different CNNs (1000 trials).

CNN	Optimistic (μ,σ)	Conservative (μ,σ)	Selective (μ,σ)
LeNet	0.31, 0.02	0.41, 0.04	0.31, 0.05
MobileNet	0.50, 0.04	6.51, 0.24	0.65, 1.13
AlexNet	0.76, 0.02	1.77, 0.05	0.78, 0.13
ResNet18	0.53, 0.04	5.09, 0.23	0.60, 0.48

optimistic scheme. For the CNNs which have higher amount of data for each layer (MobileNet and ResNet18), the conservative scheme involves a significant amount of energy consumption for sending back the results after the execution of each layer. Thus, there is a better potential for energy improvement using the selective approach for these CNNs.

MDP Sensitivity Analysis: In this section, the effect of the MDP model noise on the energy values is evaluated.

- Let P be the empirical probability function based on n observations of the round trip delay $RTT_1, ..., RTT_n$. Let π_P be the optimal MDP policy based on the probability function P.
- Let P' be the probability function based on n other observations of the round trip delay $RTT'_1, ..., RTT'_n$ that might be observed when the system is operating. P' can be considered as the *noisy model*.
- We conducted 200K experiments, in which we generated random round trip delay values $RTT'_1, ..., RTT'_n$, and applied the policy π_P with these delay values. The net result was that the energy values when applying π_P using $RTT'_1, ..., RTT'_n$ differed at most by 5% compared to the case when π_P was applied using P, which was based on $RTT_1, ..., RTT_n$. Below are the details of the experiment.

To perform the sensitivity analysis, we added a noise to the measured values of RTT. This noise was considered to have a normal distribution $N(0, \sigma^2)$. In our experiments, the minimum and maximum measured RTT values was 6 ms and 129 ms respectively. We selected the value $\sigma = 10$ for the distribution of the noise. We calculated the probability function P' based on the data with added noise. Table 5 shows the result of applying the same policy

on the delay values from both P and P'. As it can be seen, the difference in the average energy consumption for 200k is at most 5%.

Table 5. Average energy consumption for 200k trials using a same policy for the delay values from the probability function P and P'.

CNN	π_P on probability function P	π_P on noisy probability function P'	Difference
LeNet	0.416 J	0.436 J	5%
AlexNet	1.778 J	1.817 J	2%
MobileNet	5.099 J	5.345 J	5%
ResNet18	6.515 J	6.719 J	3%

The values of 95 % confidence interval are also reported for different CNNs as shown in Table 6. The confidence interval is $[\mu - 1.96 \times \sigma/n_{sample}, \mu + 1.96 \times \sigma/n_{sample}]$ where μ and σ are mean and standard deviation. These values mean that the discounted cost falls in the mentioned interval in 95 % of experiments. As can be seen, the obtained average discounted cost value also falls in this range.

It is worthy to mention that these cost values are not comparable with the total energy consumption values since they are multiplied by the discount factor in every decision epoch to evaluate the MDP solution in the infinite horizon.

Varying Bandwidth: Another set of experiments was also conducted treating the bandwidth as a random variable, sampling it at the beginning of the execution of every layer.

The bandwidth was modeled as a Gaussian random variable $N(1, 0.2)$. The mean $\mu = 1$ was the midpoint of the range of values ($[0.4, 1.6]$) MBps, whereas the standard deviation was taken to be 0.2 to cover the range of variations. The MDP state is now described by (Δ, BW, i, d), and similar to the RTT variation, it is divided into several intervals and the parameter BW is the interval within which the observed bandwidth value belongs.

Figure 12 shows the heatmap of the layer by layer execution of the CNN, assuming varying bandwidth. The figure shows that the proposed solution accounts for these variations compared with Figure 11 where the bandwidth was fixed. Table 7 shows the corresponding average values of the energy consumption of each CNN. This data shows that the energy values in the case of varying bandwidth is higher compared with the fixed bandwidth case, as expected. This difference is higher for MobileNet since it has higher number of data outputs.

	CNN	Number of network delay intervals (r)		
		$r = 3$	$r = 5$	$r = 7$
Average Discounted Cost (J)	LeNet	0.4225	0.3904	0.3767
	AlexNet	0.8831	0.8631	0.8530
	MobileNet	0.4398	0.4274	0.4218
	ResNet18	0.3204	0.3044	0.2973
95% Confidence Interval of Discounted Cost (J)	LeNet	[0.4224, 0.4226]	[0.3903, 0.3905]	[0.3765, 0.3768]
	AlexNet	[0.8831, 0.8832]	[0.8630, 0.8631]	[0.8529, 0.8531]
	MobileNet	[0.4397, 0.4398]	[0.4273, 0.4274]	[0.4218,0.4219]
	ResNet18	[0.3203, 0.3204]	[0.3044, 0.3045]	[0.2972, 0.2973]

Table 6. MDP discounted cost values and 95% confidence interval considering discount factor for different CNN models and different number of network delay intervals.

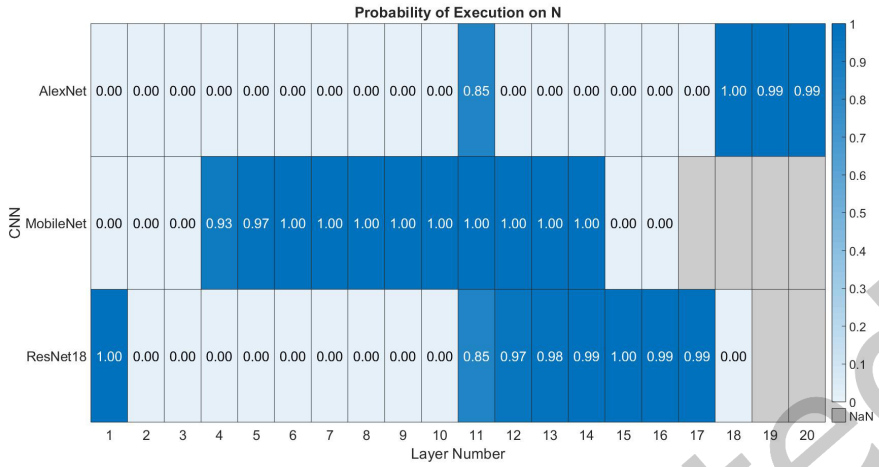


Fig. 12. The probability of executing a layer on N for different CNNs for 200k trials with having bandwidth variation.

Table 7. The average energy consumption for 200k trials considering the bandwidth variation.

CNN	Energy (Varying bandwidth) (J)	Energy (Fixed bandwidth) (J)
AlexNet	1.86	1.778
MobileNet	8.727	6.515
ResNet	5.832	5.099

7.2 Coarse-Grained Solution

The coarse-grained solutions for the case of fixed $RTT = 40ms$ are listed in Table 8 and Table 9. Column *CGDP* shows the results of executing Algorithm 2 (optimal layer partitioning between D and N), while columns *All-RPI* and *All-Jetson* refer to running the entire application either on D or on N, respectively.

Table 8. Coarse-grained solution: Energy consumption of D (in Joules) for different CNNs for the fixed network delay (RTT) of 40 *ms*. The ratio of *All-Jetson* to *CGDP* is shown in the columns.

	Conservative			Optimistic			
CNN	CGDP	All-Jetson	Ratio	CGDP	All-Jetson	Ratio	All-RPI
LeNet	0.54 J	0.63 J	1.17	0.34 J	0.34 J	1.0	0.95 J
MobileNet	7.40 J	9.1 J	1.23	0.64 J	1.96 J	3.06	20.69 J
AlexNet	1.96 J	3.10 J	1.58	0.89 J	0.89 J	1.0	2.17 J
ResNet18	5.92 J	7.84 J	1.32	0.68 J	2.07 J	3.04	6.59 J

It is worthwhile to note that in all CNNs, the *CGDP* solution outperforms the *All-RPI* and *All-Jetson* scenarios in terms of both delay and energy. MobileNet has the highest energy consumption among the networks, while LeNet has the lowest. In LeNet and MobileNet, the *All-Jetson* scenario results in lower energy consumption when compared to the *All-RPI* scenario when using the conservative data transfer scheme. Due to the larger number of layers in AlexNet and ResNet18 and the higher overhead of sending the output data in each layer back to the IoT device, the *All-RPI* outperforms *All-Jetson*. The reason that *All-RPI* shows significantly higher

Table 9. Coarse-grained solution: Completion time (in ms) for different CNNs for the fixed network delay (*RTT*) of 40 *ms*. The ratio of *All-Jetson* to *CGDP* is shown in the columns.

	Conservative			Optimistic			
CNN	CGDP	All-Jetson	Ratio	CGDP	All-Jetson	Ratio	All-RPI
LeNet	357 ms	421 ms	1.18	225 ms	225 ms	1.0	528 ms
MobileNet	2291 ms	2783 ms	1.21	199 ms	600 ms	3.01	5371 ms
AlexNet	1228 ms	2073 ms	1.69	556 ms	598 ms	1.07	1319 ms
ResNet18	1617 ms	2397 ms	1.48	208 ms	632 ms	3.04	1586 ms

energy consumption in the case of MobileNet is due to the fact that the execution time on the RPI is significantly higher. An average improvement of 31% and 23% in energy consumption was achieved compared with *All-RPI* and *All-Jetson*.

When the intermediate results are not sent back from *N* to *D* after every layer (optimistic scheme), the energy consumption of *D* for LeNet, MobileNet, AlexNet, and ResNet was significantly lower than the optimal *CGDP* solution under the conservative scheme. However, the drawback of using the optimistic scheme is that any network connection failures will entail re-computation of lost data (consuming extra energy and time).

Figure 13 shows the energy-optimal offloading solutions for different fixed *RTT* values. For LeNet, with the *RTT* of 10 *ms*, the first four layers were executed on *N* and the last two layers were computed on *D*. This is due to the fact that the computation time of last two layers in LeNet was much shorter than the communication delay. For larger values of *RTT*, the tendency to perform the computations on *D* increases. The offloading solution in LeNet is the same for three cases of delay (i.e., 20, 40, 80 *ms*). In the case of 120 *ms*, all layers are scheduled to be executed on *D*.

For AlexNet, the optimal partitioning solution is different for all five values of *RTT*. As expected, the tendency is to run more layers on *D* for larger values of *RTT*. With a network delay value of 120 *ms*, all the layers are executed on *D*. Also, it is worth mentioning that the first three layers are executed on *D* for all *RTT* values. This is because the overhead of sending data to *N* is greater than the cost of executing those layers on *D* itself.

In MobileNet, the offloading solution does not change significantly as the *RTT* changes. The first convolution layer in MobileNet takes less time to finish on *D* when compared with the intermediate convolution layers. The last two layers (pooling and fully-connected layers) also take a small amount of time to finish on *D* (less than 10

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	Energy of *D* (J)	Completion time (ms)
10	N	N	N	N	D	D	0.36	240
20	N	N	N	D	D	D	0.42	277
40	N	N	N	D	D	D	0.54	357
80	N	N	N	D	D	D	0.78	517
120	D	D	D	D	D	D	0.95	528

(a) LeNet

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	l_{19}	l_{20}	Energy of *D* (J)	Completion time (ms)
10	D	D	D	N	D	D	N	N	N	N	N	D	D	D	D	D	D	N	N	N	1.63	1028
20	D	D	D	N	D	D	D	D	D	D	N	D	D	D	D	D	D	N	N	N	1.77	1108
40	D	D	D	D	D	D	D	D	D	D	N	D	D	D	D	D	D	N	N	N	1.96	1228
80	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	N	D	D	2.11	1308
120	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	2.16	1319

(b) AlexNet

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	Energy of *D* (J)	Completion time (ms)
10	D	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	6.03	1871
20	D	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	6.49	2011
40	D	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	7.40	2291
80	D	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	9.23	2851
120	D	D	D	N	N	N	N	N	N	N	N	N	N	N	D	D	10.08	3259

(c) MobileNet

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	Energy of *D* (J)	Completion time (ms)
10	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	4.45	1376
20	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	5.09	1556
40	N	D	D	D	D	D	D	D	D	D	N	N	N	N	N	N	N	D	5.92	1617
80	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	6.59	1586
120	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	6.59	1586

(d) ResNet

Fig. 13. Energy-optimal partitioning result for CNNs in the case of different fixed network delays (Blue: Nearby device on the network (*N*), White: IoT Device (*D*)). The computed results on *N* are sent back to *D* (Conservative Scheme).

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	Energy of D (J)	Completion time (ms)
10	N	N	N	D	D	D	0.36	237●
20	N	N	N	D	D	D	0.42	277
40	N	N	N	D	D	D	0.54	357
80	N	D	D	D	D	D	0.84	515●
120	D	D	D	D	D	D	0.95	528

(a) LeNet

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	l_{19}	l_{20}	Energy of D (J)	Completion time (ms)
10	D	D	D	N	D	D	N	N	N	N	N	D	D	D	D	D	D	N	N	N	1.63448	1028
20	D	D	D	N	D	D	D	D	D	D	N	D	D	D	D	D	D	N	N	N	1.77023	1108
40	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	N	N	N	1.96273	1215●
80	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	N	D	D	2.11369	1308
120	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	2.16594	1319

(b) AlexNet

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	Energy of D (J)	Completion time (ms)
10	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	6.05	1855●
20	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	6.54	2005●
40	D	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	7.40	2291
80	D	D	D	N	N	N	N	N	N	N	N	N	N	N	D	D	9.28	2779●
120	D	D	D	N	N	N	N	N	N	N	N	N	N	N	D	D	10.08	3259

(c) MobileNet

RTT (ms)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	Energy of D (J)	Completion time (ms)
10	N	D	D	D	D	D	D	D	D	D	N	N	N	N	N	N	N	D	4.94	1317●
20	D	D	D	D	D	D	D	D	D	D	N	N	N	N	N	N	N	D	5.53	1417●
40	D	D	D	D	D	D	D	D	D	D	D	D	D	N	N	N	N	D	6.25	1569●
80	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	6.59	1586
120	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	6.59	1586

(d) ResNet

Fig. 14. Delay-optimal partitioning result for CNNs in the case of different fixed network delays (Blue: Nearby device on the network (N), White: IoT Device (D)). The computed results on N are sent back to D (Conservative Scheme).

ms). Thus, the three aforementioned layers always end up being executed on D. The solution is the same for the RTT values of 10, 20, 40, 80 *ms*. For $RTT = 120$ *ms*, the solution changes, with the second and third layers also being computed on D.

The offloading strategy is more sensitive to network delay in ResNet18 than in MobileNet. This is because each individual block in ResNet18 is less compute-intensive than the layers in MobileNet. With the RTT above 80 *ms*, all the layers are assigned to D.

Table 10 shows the outcome of applying the selective scheme in the coarse-grained case. In comparison to Table 4 for the fine-grained case, one can observe the same qualitative relationships among the three data transfer schemes under consideration. Note that Table 4 also confirms the quantitative benefit of employing the fine-grained approach over the coarse-grained approach (the numbers in Table 10 indicate higher energy consumption of D).

Table 10. Coarse-grained solution: Mean (μ) and standard deviation (σ) of energy consumption of D (in Joules) for different CNNs (1000 trials).

CNN	Optimistic (μ,σ)	Conservative (μ,σ)	Selective (μ,σ)
LeNet	0.31, 0.02	0.43, 0.77	0.32, 0.07
MobileNet	0.52, 0.08	6.59, 0.59	0.76, 1.51
AlexNet	0.80, 0.04	1.78, 0.10	0.83, 0.17
ResNet18	0.56, 0.08	5.21, 0.45	0.65, 0.56

Similar to Figure 13, which shows the energy-optimal layer assignments, Figure 14 shows the delay-optimal layer assignments. They are generated using delays as an alternative interpretation of our $E_D(\cdot)$ quantities discussed in Section 3. The cases marked with a dot indicate that the energy-optimal and delay-optimal solutions differ. As expected, the value of completion time in a delay-optimal solution is lower than in the corresponding energy-optimal case. The difference between completion times of the energy-optimal and delay-optimal assignments is larger for ResNet18. This is due to the fact that the difference between computation and communication power is higher for this CNN. The results presented here demonstrate again that the network delay can have a significant impact on the optimality of different layer assignments.

Figure 15 shows the results of energy-optimal assignment where the communication bandwidth was changed from 1.5 MBps to 1 MBps for $RTT = 20ms$. When the bandwidth is decreased, the tendency to run on D increases.

For instance, layers l_2 and l_3 of MobileNet were executed on N in the case of $B = 1.5MBps$ where these layers were executed on D when the bandwidth was decreased. In all cases, the energy consumption and delay were increased when the bandwidth was decreased.

B (MBps)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	Energy of D (J)	Completion time (ms)
1.5	D	N	N	N	N	N	N	N	N	N	N	N	N	N	D	D	6.49	2011
1	D	D	D	N	N	N	N	N	N	N	N	N	N	N	D	D	9.70	2907
0.5	D	D	D	D	D	N	N	N	N	N	N	N	N	N	D	D	13.31	3870

MobileNet

B (MBps)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	Energy of D (J)	Completion time (ms)
1.5	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D	5.09	1556
1	D	D	D	D	D	D	D	D	D	D	N	N	N	N	N	N	N	D	5.84	1513
0.5	D	D	D	D	D	D	D	D	D	D	D	D	D	N	N	N	N	D	6.45	1620

ResNet18

B (MBps)	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}	l_{15}	l_{16}	l_{17}	l_{18}	l_{19}	l_{20}	Energy of D (J)	Completion time (ms)
1.5	D	D	D	N	D	D	D	D	D	D	N	D	D	D	D	D	D	N	N	N	1.77	1108
1	D	D	D	D	D	D	D	D	D	D	N	D	D	D	D	D	D	N	N	N	1.86	1160
0.5	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	N	N	N	1.90	1171

AlexNet

Fig. 15. Energy-optimal partitioning result for CNNs in the case of different bandwidth values (Blue: Nearby device on the network (N), White: IoT Device (D)). The computed results on N are sent back to D (Conservative Scheme).

7.3 Comparison with Related Work

As mentioned in prior work section, the previous papers have not dealt with the computation offloading on low-end IoT devices considering reliability of communication channel and stochastic communication delays. However, in this subsection, our partitioning solution has been compared with two previous papers [13, 22] proposed for high-end mobile devices. It is shown that our approach leads to optimal results with having significantly lower overhead of decision making.

The solution presented in [22] finds the partition point in the layers of a CNN. As the result of partitioning algorithm, the layers before the partition point are mapped onto IoT device and the rest of layers are scheduled on the nearby accelerator. This style of partitioning might not lead to the optimal solution since there might be interleaving of layers as shown in Figure 13. Table 11 shows the energy consumption of [22] compared with our solution for four CNNs in the experiments for 1000 frames using conservative data transfer scheme. The energy saving is dependent on the complexity of layers and the possible interleaving of layers. On average, 17% improvement was obtained using our approach compared with the work [22].

Table 11. Energy consumption of [22] compared with our approach for 1000 frames. Our approach leads to an average of 17% improvement for four CNNs compared with [22].

CNN	EdgeWise (Proposed solution)	Ref. [22]	Improvement
LeNet	0.43 J	0.47 J	9%
AlexNet	1.78 J	1.92 J	8%
MobileNet	6.59 J	8.17 J	24%
ResNet18	5.12 J	6.53 J	27%

Figure 16 shows the empirical CDF of the energy consumption for EdgeWise (the proposed approach) compared with the presented method in [22] for MobileNet CNN. As it can be seen, the method in [22] will lead to higher energy consumption compared with EdgeWise.

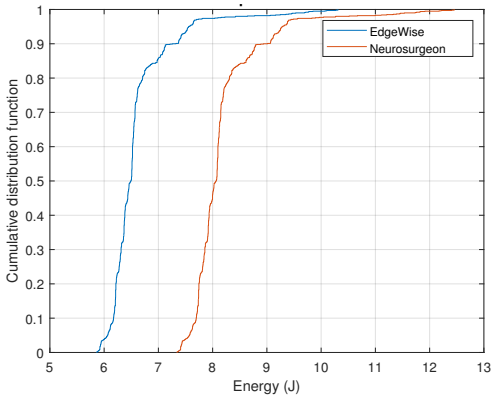


Fig. 16. EdgeWise leads to lower energy consumption compared with Neurosurgeon [22] as shown in the empirical CDF for MobileNet.

Our partitioning solution was also compared with the method presented in [13]. A solution based on finding the shortest path is presented in [13]. Table 12 shows the overhead of decision making for our proposed solution and the shortest path method proposed in [13]. As it can be seen, our approach has significantly lower overhead due to the O(1) complexity compared with the shortest path approach.

The solution presented in [13] does not consider the reliability of communication channel. The focus of our work is to provide solutions for tackling unreliable communication delays where the assignment of layers to devices is determined based on the prediction of network status. Data transfer scheme is selected based on the prediction of the network behavior to ensure that the computation is not done from the beginning in the case of unavailability of N. The presented MDP solution considers the stochastic behavior of network in the middle of processing a frame and uses a lookup table to select the optimal action in each state of the system.

Table 12. Comparison of decision making overhead for our proposed approach versus the shortest path solution in [13]. Our solution has significantly lower overhead of decision making.

CNN	EdgeWise	Shortest Path	Speedup over Shortest Path
LeNet	21 μs	0.6 ms	28
MobileNet	41 μs	1.2 ms	29
AlexNet	50 μs	1.5 ms	30
ResNet18	45 μs	1.3 ms	28

8 CONCLUSIONS

This paper addressed the problem of energy-efficient execution of CNNs on an energy-constrained embedded IoT device in collaboration with a nearby accelerator on the wireless network in the presence of stochastic communication delays. The problem of minimizing the energy consumption of an IoT device has been addressed using fine-grained and coarse-grained approaches. In the fine-grained approach, where the network delay is sampled before executing any given layer, the problem is formulated and solved as a Markov Decision Process (MDP) that assigns the optimal action (i.e., deciding where to execute that layer) in each state of MDP based on a lookup table generated offline. During run-time, the lookup table is accessed and the corresponding action related to the current state is determined in $O(1)$ time. In the coarse-grained approach where the network delay

is sampled only once at the beginning (and then assumed fixed during the CNN application execution), the problem is formulated and solved via dynamic programming of linear time complexity. It should be noted that for both approaches, the generated solutions are not only lightweight, but also optimal. The proposed solutions can be also used for the case of multiple nearby accelerators where the time complexity of both fine-grained and coarse-grained solutions remains the same.

The experimental results for four CNNs, executed on a Raspberry Pi (IoT device) in collaboration with an NVIDIA Jetson (accelerator device), show that: (1) significant energy savings can be achieved using our proposed methods in comparison to the no-collaboration case, and (2) the stochastic fine-grained approach yields larger energy savings in comparison to the deterministic coarse-grained approach.

ACKNOWLEDGMENTS

This research was supported in part by NSF Grant Numbers 2008244, CCF-1652132 and CCF-1618039 and by the Center for Embedded Systems, NSF Grant 1361926.

REFERENCES

[1] 2020. NVIDIA Jetson TX2. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.
[2] 2020. Raspberry Pi 3 Model B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.
[3] 2020. UniFi AP AC LITE. https://www.ui.com/unifi/unifi-ap-ac-lite/.
[4] Frank Adelstein, Sandeep KS Gupta, Golden Richard, and Loren Schwiebert. 2005. *Fundamentals of Mobile and Pervasive Computing*. Vol. 1. McGraw-Hill New York.
[5] Hossein Badri, Tayebeh Bahreini, Daniel Grosu, and Kai Yang. 2019. Energy-Aware Application Placement in Mobile Edge Computing: A Stochastic Optimization Approach. *IEEE Transactions on Parallel and Distributed Systems* (2019).
[6] Erfan Bank-Tavakoli, Seyed Abolfazl Ghasemzadeh, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2019. POLAR: A Pipelined/Overlapped FPGA-Based LSTM Accelerator. *IEEE Transactions on VLSI* (2019).
[7] Richard Bellman. 1957. A Markovian decision process. *Journal of Mathematics and Mechanics* (1957), 679–684.
[8] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 155–168.
[9] Mung Chiang and Tao Zhang. 2016. Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal* 3, 6 (2016), 854–864.
[10] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, 49–62.
[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Conference on Computer Vision and Pattern Recognition*. 248–255.
[12] Koustabh Dolui and Soumya Kanti Datta. 2017. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *Global Internet of Things Summit (GIoTS), 2017*. IEEE, 1–6.
[13] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* (2019).
[14] Alan Ferrari, Silvia Giordano, and Daniele Puccinelli. 2016. Reducing your local footprint with anyrun computing. *Computer Communications* 81 (2016), 1–11.
[15] C. M. Fiduccia and R. M. Mattheyses. 1982. A Linear-Time Heuristic for Improving Network Partitions. In *Proceedings of the Design Automation Conference*. 175–181.
[16] Wei Gao, Yong Li, Haoyang Lu, Ting Wang, and Cong Liu. 2014. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *22nd International Conference on Network Protocols (ICNP)*. IEEE, 1–12.
[17] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. 2020. The architectural implications of facebook's DNN-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 488–501.
[18] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards wearable cognitive assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 68–81.
[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*. 770–778.

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
[21] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. 2018. Computation Offloading for Machine Learning Web Apps in the Edge Server Environment. In *International Conference on Distributed Computing Systems*. IEEE, 1492–1499.
[22] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *ACM SIGARCH Computer Architecture News*, Vol. 45. ACM, 615–629.
[23] Mahir Kaya, Altan Koçyiğit, and P Erhan Eren. 2016. An adaptive mobile cloud computing framework using a call graph based model. *Journal of Network and Computer Applications* 65 (2016), 12–35.
[24] Young Geun Kim and Carole-Jean Wu. 2020. AutoScale: Energy Efficiency Optimization for Stochastic Edge Inference Using Reinforcement Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1082–1096.
[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.
[26] Hanna Kurniawati, David Hsu, and Wee Sun Lee. 2008. SARSOP: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, Vol. 2008. Zurich, Switzerland.
[27] Yann LeCun. 2015. LeNet-5, convolutional neural networks (online). http://yann. lecun. com/exdb/lenet. Accessed: 2019-03-12.
[28] En Li, Zhi Zhou, and Xu Chen. 2018. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications*. 31–36.
[29] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Network* 32, 1 (2018), 96–101.
[30] Xiaoqiang Ma, Tai Yao, Menglan Hu, Yan Dong, Wei Liu, Fangxin Wang, and Jiangchuan Liu. 2019. A survey on deep learning empowered IoT applications. *IEEE Access* 7 (2019), 181721–181732.
[31] Jiachen Mao, Xiang Chen, Kent W Nixon, Christopher Krieger, and Yiran Chen. 2017. Modnn: Local distributed mobile computing system for deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 1396–1401.
[32] Roberto Fernandez Molanes, Kasun Amarasinghe, Juan Rodriguez-Andina, and Milos Manic. 2018. Deep Learning and Reconfigurable Platforms in the Internet of Things: Challenges and Opportunities in Algorithms and Hardware. *IEEE Industrial Electronics Magazine* 12, 2 (2018), 36–49.
[33] Sylvie Ong, Shao Png, David Hsu, and Wee Lee. 2009. POMDPs for robotic tasks with mixed observability. (2009).
[34] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *Conference on Neural Information Processing Systems*.
[35] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. 2017. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078* (2017).
[36] Olivier Sigaud and Olivier Buffet. 2013. *Markov Decision Processes in Artificial Intelligence*. John Wiley & Sons.
[37] Nikolai Smolyanskiy et al. 2017. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. In *IROS*. IEEE, 4241–4247.
[38] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *International Symposium on Field-Programmable Gate Arrays*. ACM, 16–25.
[39] Rahul Urgaonkar, Shiqiang Wang, Ting He, Murtaza Zafer, Kevin Chan, and Kin K Leung. 2015. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation* 91 (2015), 205–228.
[40] Siqi Wang, Gayathri Ananthanarayanan, Yifan Zeng, Neeraj Goel, Anuj Pathania, and Tulika Mitra. 2019. High-throughput cnn inference on embedded arm big. little multi-core processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019).
[41] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K Leung. 2015. Dynamic service migration in mobile edge-clouds. In *IFIP Networking Conference (IFIP Networking)*. IEEE, 1–9.
[42] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. 2018. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 1–12.
[43] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. 2018. DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2348–2359.