In-situ AI: Towards Autonomous and Incremental Deep Learning for IoT Systems

Mingcong Song*, Kan Zhong†*, Jiaqi Zhang*, Yang Hu‡*, Duo Liu†, Weigong Zhang§, Jing Wang§ and Tao Li*
*University of Florida, †Chongqing University, ‡University of Texas at Dallas, §Capital Normal University
{songmingcong, jiaqizhang}@ufl.edu, kzhong1991@cqu.edu.cn, taoli@ece.ufl.edu

***Abstract*—Recent years have seen an exploration of data volumes from a myriad of IoT devices, such as various sensors and ubiquitous cameras. The deluge of IoT data creates enormous opportunities for us to explore the physical world, especially with the help of deep learning techniques. Traditionally, the Cloud is the option for deploying deep learning based applications. However, the challenges of Cloud-centric IoT systems are increasing due to significant data movement overhead, escalating energy needs, and privacy issues. Rather than constantly moving a tremendous amount of raw data to the Cloud, it would be beneficial to leverage the emerging powerful IoT devices to perform the inference task. Nevertheless, the statically trained model could not efficiently handle the dynamic data in the real in-situ environments, which leads to low accuracy. Moreover, the big raw IoT data challenges the traditional supervised training method in the Cloud. To tackle the above challenges, we propose In-situ AI, the first Autonomous and Incremental computing framework and architecture for deep learning based IoT applications. We equip deep learning based IoT system with *autonomous IoT data diagnosis* (minimize data movement), and *incremental and unsupervised training method* (tackle the big raw IoT data generated in ever-changing in-situ environments). To provide efficient architectural support for this new computing paradigm, we first characterize the two In-situ AI tasks (i.e. inference and diagnosis tasks) on two popular IoT devices (i.e. mobile GPU and FPGA) and explore the design space and tradeoffs. Based on the characterization results, we propose two working modes for the In-situ AI tasks, including Single-running and Co-running modes. Moreover, we craft analytical models for these two modes to guide the best configuration selection. We also develop a novel *two-level weight shared In-situ AI architecture* to efficiently deploy In-situ tasks to IoT node. Compared with traditional IoT systems, our In-situ AI can reduce data movement by 28-71%, which further yields 1.4X-3.3X speedup on model update and contributes to 30-70% energy saving.**

I. Introduction

Recently, Internet of Things (IoT) technology, which connects numerous physical things to the Internet, has radically increased our ability to sense data from the physical world (e.g. buildings, factories, automobiles, the environment, and even ourselves). Networking giant Cisco estimates that the number of connected devices worldwide will be up to 50 billion in 2020 [1]. The resulting deluge of data from these IoT devices creates enormous opportunities for us to explore the physical world, leading towards a smart life. However, realizing this potential requires us to address a huge challenge — extracting meaningful information and patterns based on the raw IoT data captured from noisy and complex environments. Deep Learning, one of the most promising approaches, can perform more robust and reliable inference tasks. To achieve this, the deep learning techniques leverage many-layer neural networks to learn levels of representation and abstraction that make sense of data. Recently, deep learning based approaches have achieved great success on many IoT applications, such as smart cities [2], transportation [3], and smart farming [4].

Traditionally, due to its huge compute power and scalability, the Cloud is often the best option for training and evaluating neural networks. For Cloud-centric IoT systems, data generated from an IoT device first needs to be sent to the Cloud for processing, after that the result is sent back to the IoT device. However, the large data transmission through networks presents a grand challenge and may raise privacy issues as well.

Recently, with the increasing compute power of mobile devices, there is a growing interest in performing deep learning inference task on mobile platforms [5–7]. The idea is similar to the in-situ data processing scheme called fog computing proposed by Cisco [8]. Although in-situ/fog computing eliminates the effect of fluctuating network quality and protects users privacy (sensitive data is processed locally), it introduces a new problem: low accuracy in the inference results. In fact, many deep learning based studies have relied on the same assumption: the world hands me a static, potentially very large, ideal dataset and I train an accurate, potentially complex model [9]. This fiction departs from the real IoT systems in two key factors: *data is dynamic and unlabeled*. Therefore, the statically trained model could not efficiently handle the dynamic data in the real in-situ environments. For example, in wild animal monitoring, the changing environmental conditions, such as variation in lighting, could affect the image quality and pose a great challenge towards achieving acceptable accuracy. To achieve an ideal accuracy, all the real IoT data still needs to be transferred to the Cloud to retrain the model, which results in the same challenges with the Cloud-centric IoT system: big data movement, requirement of stable and fast network and privacy issues. This motivates us to design a method to *improve the accuracy of IoT system with the minimum data movement*. Moreover, the gathered IoT data is largely unlabeled and it is impractical to have human to label all the IoT raw data. Therefore, the traditional supervised training is not suitable for the IoT based system, which demands *an unsupervised method to truly unleashing the potential of big raw IoT data*.

2378-203X/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCA.2018.00018

To tackle the above challenges, we propose In-situ AI, an Autonomous and Incremental computing framework and architecture for deep learning based IoT applications. Our In-situ AI consists of two parts: In-situ AI Node and Cloud. In the Cloud, we first utilize *an unsupervised pre-training method* to obtain the features from big raw IoT data. Then the inference network can be trained based on the features learned by the unsupervised network (we name this procedure as *transfer learning*). Since the unsupervised pre-training is well-trained on the big raw IoT data, the inference network, which is based on the unsupervised network, can *achieve a high accuracy using a limited amount of labeled data*. Our In-situ AI node features a novel function: autonomous IoT data diagnosis (diagnosis task). It allows the IoT nodes to detect whether the IoT data is valuable (unrecognized) or not; and send the valuable data to the Cloud for incremental training. With the diagnosis task, only a small proportion of IoT data (i.e., incorrect predictions) needs to be uploaded to the Cloud for further action. Note that our In-situ AI Node is an edge-computing node, which is responsible for processing data from multiple sensors. By incrementally training the deep learning models on the incoming IoT data, our In-situ AI can *fit the ever-changing environments in the in-situ system with reduced data movement*.

In an IoT node of In-situ AI, both inference task and diagnosis task are running simultaneously, which brings tremendous computing pressure for IoT devices. To efficiently deploy these two tasks on IoT devices, we first characterize these In-situ AI tasks (i.e. inference and diagnosis) on two popular IoT devices (i.e. mobile GPU and FPGA). Our characterization results show: (1) a tradeoff exists between fast response time and energy-efficiency; (2) latency and energy-efficiency are two key metrics for inference task, while energy-efficiency is the only design concern for the diagnosis task (3) GPU's energy-efficiency is always better than that of FPGA when only one AI task is running, while the interference on GPU is severe when two AI tasks are concurrently running; (4) a weight shared architecture is demanded for FPGA implementation. Then we propose two working modes (i.e. Single-running and Co-running) to meet the various needs of IoT environments. The Single-running mode focuses on the situation where the inference task is not always running. Thus, diagnosis task can share the same in-situ platform with inference task at different time slots. The Co-running mode is applied to the scenario where the inference task should be available 24/7. In this mode, diagnosis task has to run with inference task simultaneously on the in-situ platform. We craft various analytical models, including time and resource models, to guide the best configuration selection for these two working modes. To avoid the interference between inference and diagnosis tasks in Co-running mode, we implement a two-level weight shared FPGA design.

Figure 1: Two Typical Deep Learning based IoT Systems.

Figure 2: Samples from Real IoT Data [12].

Evaluation results show that our analytical model could effectively guide the configuration selection across various IoT scenarios. Compared with traditional IoT systems, our In-situ AI can reduce data movement by 28-71%, which further yields 1.4X-3.3X speedup on model update and contributes to 30-70% energy saving.

In summary, we make the following key contributions:

- We present In-situ AI, the first Autonomous and Incremental in-situ computing framework and architecture for deep learning based IoT applications, to improve the accuracy of traditional deep learning based IoT systems with minimum data movement.
- To the best of our knowledge, this paper is the first work to introduce incremental and unsupervised learning in IoT system to tackle the big raw IoT data generated in ever-changing in-situ environments.
- Our In-situ AI features an autonomous IoT data diagnosis, which greatly reduces the communication overhead and the pressure of model updating in the Cloud.
- We develop a novel two-level weight shared architecture for In-situ AI tasks and craft analytical models to guide the best configuration selection.

The rest of this paper is organized as follows. Section II introduces the background and motivation. Section III proposes our In-situ AI framework. Section IV describes our In-situ AI architecture design. Section V evaluates the In-situ AI. Related works and conclusions are discussed in Section VI and VII, respectively.

II. Background and Motivatoin

A. Deep Learning based IoT System: The Challenges

1) Cloud-centric IoT System: Fig. 1(a) illustrates a typical Cloud-centric IoT system. The raw data acquired by the sensors (e.g. cameras) in IoT nodes is sent to neural networks deployed in the Cloud. After the inference task (e.g. classification and recognition) is performed, the result is sent back to the IoT node. However, there are several disadvantages for this Cloud-centric IoT system: (1) a large amount of data is uploaded to the Cloud via networks, resulting in energy and latency overheads; (2) a fast and stable connection to the Cloud is mandatory at all times, which is impractical in many scenarios, such as astronomy observation in a remote area, video surveillance for wildlife

TABLE I: ACCURACY OF CNN MODELS ON SERENGETI

Data set	AlexNet (80%)	GoogleNet (83%)	VGGNet (93%)
Serengeti	54%	62%	72%

behavioral studies; and (3) sending sensor data to the Cloud may raise security and privacy issues.

2) In-situ/Fog Computing: To tackle the challenges in Cloud-centric IoT system, Fig. 1(b) introduces in-situ/fog computing based IoT system, where the deep learning model is trained in the Cloud using a static, potentially very large, ideal dataset, such as ImageNet [10]. Then the model is deployed into the IoT node to perform the inference task.

However, since real IoT data is collected in a dynamic and less ideal condition, there is a new challenge in the In-situ based IoT system: low accuracy in the inference results. For example, the pictures in Fig. 2 are selected from Snapshot Serengeti dataset [11], which consists of millions of real camera trap images in Serengeti National Park. In the real in-situ environment, animals do not behave in a predictable way like Fig. 2(a). For instance, some images do not contain the whole body of an animal because the animal is too close to the camera (see Fig. 2(b)). Also, animals may be captured in random poses as Fig. 2(c). Weather condition also affects the quality of images, such as poor illumination in Fig. 2(d). TABLE I illustrates the accuracy of various Convolutional Neural Networks (CNNs) [12–14] on these real IoT data (Snapshot Serengeti Dataset). As shown, all the well-trained CNNs on large-scale ImageNet dataset suffer a much lower accuracy in real scenarios. For instance, the accuracy of AlexNet drops from 80% to 54%. Therefore, the static model trained in the Cloud could not efficiently handle the dynamic data in the real in-situ environments.

To cater to the ever-changing environment in in-situ systems and achieve an ideal accuracy, these deep learning models should be incrementally trained on the real IoT data. Since the training procedure is performed in the Cloud, all the real IoT data needs to be transferred to the Cloud, which still results in the same challenges with the Cloud-centric IoT system: big data movement, requirement of stable and fast network and privacy issues. This motivates us to design a method *to improve the accuracy of IoT system with the minimum data movement.*

Traditionally, the deep learning models are trained in the Cloud with a supervised method, which requires a large amount of human efforts to label the training data. In the IoT system, with more and more raw data from numerous IoT nodes, it is difficult for us to label these big IoT data. Therefore, the traditional supervised training is not suitable for the IoT based System, which demands an *unsupervised method to truly unleashing the potential of big raw IoT data.*

III. In-situ AI Framework

In this section, we first provide an overview of the proposed In-situ AI framework. Then we evaluate our system by examining whether it solves the challenges in the traditional deep learning based IoT systems. Finally, we analyze the characteristics of the tasks running in our In-situ AI.

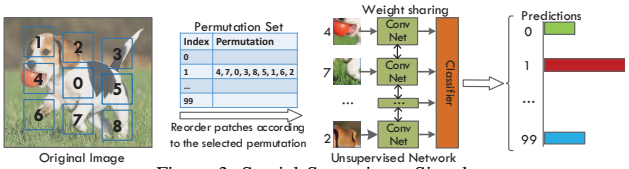

Figure 3: Spatial Supervisory Signal.

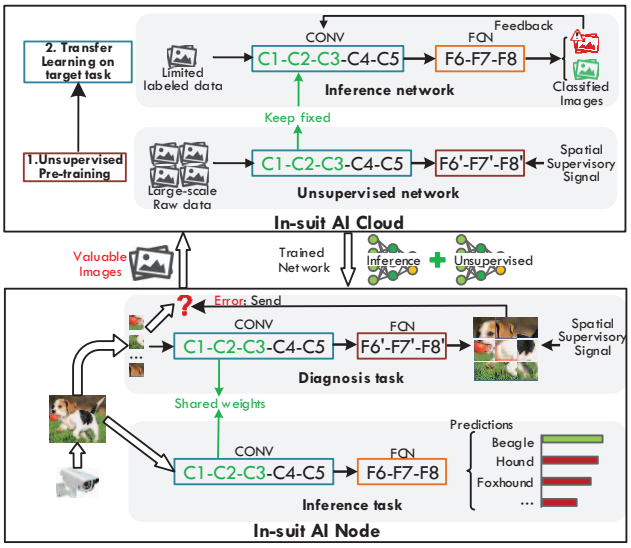

Figure 4: An Overview of In-situ AI Framework.

A. An Overview of In-situ AI

To tackle the challenges of big raw IoT data, we argue that the deep learning based IoT system should be equipped with an unsupervised method and improve accuracy with reduced data movement. In this subsection, we first introduce how to implement the unsupervised learning method. Then, we give an overview of our In-situ AI framework. Finally, we discuss how to improve the accuracy of IoT system with the minimum data movement.

Although it is impractical to label all the IoT data, fortunately there are some supervisory signals in IoT data that we can leverage to perform the unsupervised training. Spatial context information is one of such supervisory signals [15–17]. Fig. 3 demonstrates how to leverage spatial context information (the relative positions of patches within an image) to implement unsupervised training [15]. First, we divide an image into a 3x3 grid. These 9 tiles are reordered via a randomly chosen permutation ([4,7,0,3,8,5,1,6,2] in Fig. 3) from a predefined permutation set and are then fed into the unsupervised neural network. The task is to predict the index of the chosen permutation. Its output is a probability vector with 100 classes and each class corresponds to one index in the permutation set. After the network is well-trained, the class with the same index as the chosen permutations index should have the max probability (Class1 in Fig. 3), which means the network could correctly

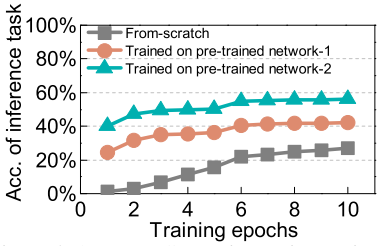

Figure 5: Accuracy Comparison using Various Training Methods.

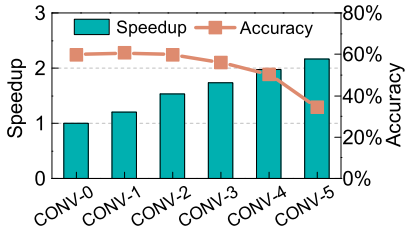

Figure 6: Accuracy and Time Comparisons by Fine-tuning Different Layers.

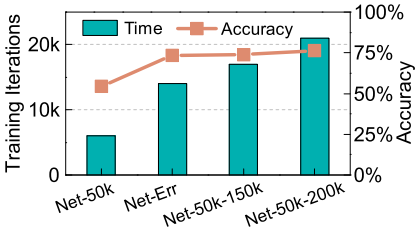

Figure 7: Unsupervised pre-training on Datasets with Different Sizes.

recognize the relative positions of 9 tiles. Recent research demonstrates that doing well on this context prediction task requires the deep learning model to be sophisticated at recognizing objects and their parts [17], meaning this task is related to the object recognition task [15]. Therefore, the features learned from this unsupervised context prediction task can be further leveraged to improve the accuracy of recognition task.

Fig. 4 illustrates our proposed In-situ AI framework, which consists of In-situ AI node and Cloud. In the Cloud, we first train the unsupervised network, which leverages the spatial context information to obtain features of objects from the big raw IoT data, and this procedure is named as *unsupervised pre-training*. Then, the extracted features are transferred to implement a target inference network, such as object recognition. As we know, the deep convolutional neural network consists of many convolutional layers, where its first layers learn features similar to Gabor filters and color blobs [18]. Such first-layer features appear not to be specific to a particular dataset or task, but general in that they are applicable to many datasets and tasks. Therefore, our transfer learning approach is to train an unsupervised network using the method in Fig. 3 and then copy its first n layers (n is 3 in Fig. 4) to the first n layers of a target inference network. In this way, the inference network could leverage the existing features in the unsupervised network and it only needs to train the remaining layers with a limited amount of labeled data. In Fig. 4, the inference network performs the transfer learning based on the unsupervised network, so the accuracy of inference network is proportional to the accuracy of the unsupervised network. Since the unsupervised pre-training is well-trained on the big raw IoT data, the inference task, which is based on the unsupervised network, can *achieve a high accuracy using a limited amount of labeled data*.

To cater to the ever-changing environment in in-situ systems, the network needs to be incrementally trained on the incoming IoT data. Traditionally, we utilize all IoT raw data to retrain the network. However, not all the raw data have the same value. The training data can be divided into two classes: recognized class (could be correctly recognized) and unrecognized class (fails to be recognized). Compared to the recognized data, unrecognized data is more valuable for the accuracy of network. As shown in Fig. 4, besides the inference task running on IoT node, we deploy the unsupervised network trained in the Cloud to the IoT node to select the valuable data autonomously (we name this task as diagnosis task). At the beginning, we deploy initialized inference and diagnosis models (trained with a limited amount of data) to our In-situ AI node to perform the inference and diagnosis tasks respectively. Then, these models can be incrementally improved with more and more newly acquired IoT data. During this procedure, the diagnosis task in IoT node only sends the unrecognized data to the Cloud for further incremental training, which greatly reduces the data movement. In this way, our In-situ AI can *improve its accuracy on dynamic data in real in-situ environments with the reduced data movement*.

To summarize, by utilizing unsupervised pre-training on the large-scale raw IoT data, the accuracy of deep learning based IoT system could be greatly improved. Moreover, with the diagnosis task on IoT node, most IoT data can be processed locally and only a small proportion needs to be uploaded to the Cloud for further action.

B. Evaluation of In-situ AI Framework

In this subsection, we evaluate our proposed In-situ AI framework by answering the following questions. First, will the unsupervised pre-training improve the accuracy of the inference task? Second, how much are the unsupervised pre-trained network and inference network related? Third, although only transferring the valuable data could reduce the data movement, is this enough to improve the accuracy? In the following experiments, we utilize NVIDIA Titan X to perform the training. Our training data and testing data are from ILSVRC2012 [10]. All the deep learning models are trained by Caffe [19].

First, we validate whether the unsupervised pre-trained network improves the accuracy of inference network. Fig. 5 illustrates the accuracy comparison using various training methods. The gray line indicates the network is trained from scratch on a limited labeled data (100k images). The other two lines demonstrate the network is transfer-learned from an unsupervised pre-trained network using the same amount of labeled data (100k images). Among them, the orange line is based on a pre-trained network with 71% accuracy, while the green line corresponds to the pre-trained network with 88% accuracy. With the unsupervised pre-trained network, the accuracy of inference network could be greatly improved (30% improvement). Moreover, higher

accuracy on the unsupervised pre-trained network (green line in Fig. 5) will lead to a better recognition result for the inference network.

Next, we demonstrate how much the unsupervised pre-trained network and inference network are related. We take the unsupervised pre-trained network and transfer its features to train the inference task by locking various convolutional layers. In Fig. 6, CONV-i indicates that all layers from conv1 to convi are locked and all subsequent layers are randomly initialized and retrained. If we keep no layer locked and retrain all the layers (CONV-0), we obtain the maximum accuracy (59%). The accuracy is 34% when only fully connected layers are trained (CONV-5). There is a significant improvement (from 34% to 56%) when *conv4* and *conv5* are also trained (CONV-3). This shows that the features learned from *conv1* to *conv3* are also applicable to the inference task and they can share the weights of first three convolutional layers. Note that the weights sharing will reduce the training time since the weights in the first layers are fixed and do not need to be retrained. Compared with CONV-0 (without weight sharing), the weight sharing from *conv1* through *conv3* can achieve 1.7X speedup.

Finally, we validate whether the valuable data benefits accuracy in Fig. 7. There are total 200k images in this experiment. We take the incremental training method. First, we train a network from scratch using 50k images (Net-50k). Then we leverage this network to process the remaining 150k images and obtain the images with incorrect results. Finally, we train a network (Net-Err) by fine-tuning Net-50k on these incorrect images. We also obtain Net-50k-150k and Net-50k-200k by fine-tuning Net-50k on the remaining 150k images and all the 200k images respectively. Compared with Net-50k, our Net-Err (only using incorrect images) nearly achieves the same accuracy improvement as Net-50k-200k (using all 200k images). Since it is only fine-tuned on the incorrect images, Net-Err not only demands the minimum data movement but also takes the least time to finish the transfer learning as shown in Fig. 7.

C. An Analysis of In-situ Tasks

In this subsection, we give a brief introduction to inference and diagnosis tasks and analyze their characteristics.

1) Inference Task: In IoT systems, the inference task is mainly responsible for object detection, recognition, and classification, such as automatic animal recognition in a wildlife sanctuary and anomaly detection in video surveillance systems. These kinds of systems usually collect a large set of data and demand real-time response. Therefore, the inference task is an online task and must process the inputs as quickly as possible. Moreover, most IoT nodes are battery-powered, making them sensitive to energy consumptions. Thus, an energy-efficient inference task is quite important. To summary, in the design of inference task, *the latency and energy-efficiency* are two key metrics for optimization (i.e. consuming minimum energy under given runtime constraint).

2) Diagnosis Task: The diagnosis task is responsible for selecting the valuable data to upload to the Cloud for further action. However, diagnosis task will aggravate the processing burden in the IoT node, especially for the IoT nodes with a limited computing resource. Fortunately, there are still some opportunities to optimize the deployment of diagnosis task. First, the latency of diagnosis task is not critical since the incorrectly recognized data is not required to be sent back to the Cloud immediately. Choosing the valuable IoT data can be deferred until the computing resource is available. Therefore, *energy-efficiency* is the only design concern for the diagnosis task. Second, as shown in Fig. 4, the unsupervised network for diagnosis task shares some CONV layers with the inference network and they utilize the same weights. What is more, for the diagnosis task, all its input patches (9 patches in Fig. 3) also share the same CONV layers. These two kinds of *weight sharing* in the CONV layers provide us an opportunity to simplify the design of In-situ AI architecture.

IV. In-situ AI Architecture

To design an efficient architecture for In-situ AI, we first characterize our In-situ AI tasks on the two mainstream in-situ platforms: Mobile GPU (i.e. NVIDIA TX1 [20]) and FPGA (i.e. Xilinx Virtex-7 VX690T [21]). These devices are deployed into the edge-computing nodes, which process data from multiple sensors. In this section, we first introduce two state-of-the-art designs for deep neural networks on GPU and FPGA respectively. Based on the characterization results, we propose two working modes for the In-situ AI tasks: Single-running and Co-running modes. Then we design our In-situ AI architectures for these two modes.

A. Characterizations of In-situ AI Tasks

1) State-of-the-art Design: As we know, convolutions are the most performance-critical operations in CNNs, involving computationally intensive multiply-accumulate operations. As shown in Fig. 8, one input of CONV layers is a 3-dimension image, consisting of N 2-dimension input feature maps. The other input is a 4-dimension kernel, which has M filters and each filter has N channels of $K \times K$ weights. The output is generated by 3-dimensional convolutions of the filters with the inputs. Each filter generates one output feature map, resulting in M feature maps. Next, we introduce the state-of-the-art implementation of CONV layers on GPU and FPGA.

a) GPU: To leverage the well-optimized matrix-multiplication libraries, such as cuBLAS and cuDNN [22], GPU normally converts the convolutional operations into matrix-multiplication operations. As shown in Fig. 8, in step ①, an operation called im2col [23] stretches out the local regions in the input features into column-major

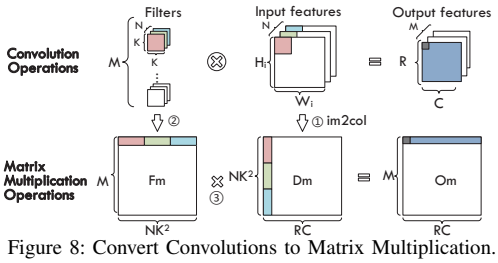

Figure 8: Convert Convolutions to Matrix Multiplication.

matrix (D_m). Similarly, in step ② the weights of the filters are stretched out into filter matrix (F_m). Then the original convolutional operation is converted into a matrix multiplication ($F_m \times D_m$) in step ③. In Fig. 8, the filter matrix F_m has dimensions $M \times NK^2$, while the data matrix D_m has dimensions $NK^2 \times RC$. The output matrix O_m has dimensions $M \times RC$. Therefore, we can calculate the number of operations (ops) in a CONV layer based on the number of multiply-accumulate operations of $F_m \times D_m$:

$$CONV_{ops} = 2 \times M \times N \times K^2 \times R \times C, \quad (1)$$

where a single multiply-accumulate operation counts as 2 ops. The $CONV_{ops}$ is usually used to measure the computational intensity in a CONV layer. The computational procedure of matrix-matrix multiplication on GPU is similar to the algorithm developed by Volkov and Demmel [24]. It divides the output matrix O_m into multiple sub-matrices of the size of $m \times n$ and each of these sub-matrices is computed by a thread block. Therefore, the number of thread blocks (denoted by $Grid_{size}$) used for computing the output matrix is determined by the number of sub-matrices:

$$Grid_{size} = \left\lceil \frac{M}{m} \right\rceil \times \left\lceil \frac{R \times C}{n} \right\rceil. \quad (2)$$

Normally, $Grid_{size}$ should be larger than the maximum number of blocks ($maxBlocks$) that could be simultaneously executed on GPU so that GPU resource is fully utilized.

b) FPGA: Note that the matrix-multiplication based convolution comes at the expense of data duplication, which diminishes the overall performance gains in bandwidth-limited FPGA platforms [25]. Moreover, the transformation (i.e. im2col) also introduces extra overheads. Therefore, the state-of-the-art implementation of CONV layers on the FPGA, such as DianNao [26] and Eyeriss [27], are still based on traditional convolutional operations.

The pseudo code of CONV layer is shown in Fig. 9. It employs loop tiling, reordering and unrolling to increase processing throughput and reduce data transfer. Since the innermost two loops are unrolled (based on T_m and T_n), it is able to process T_n input feature maps and T_m output feature maps in parallel. As shown in Fig. 10, to implement these two unrolled loops, T_m vector dot-product units (each of them has a width T_n) are constructed. An adder is added after each unit, which yields $T_m \times T_n$ multipliers. Given a resource budget (e.g. the number of DSP slices), one can find the optimal T_m and T_n for a given CONV layer.

```
for (r = 0; r < R; r += Tr)                          → Loop tiling
 for (c = 0; c < C; r += Tc)
  for (m = 0; m < M; m += Tm) {
   for (n = 0; n < N; n += Tn) {
    Load(Input);
    Load(Weight);
    for (tr = r; tr < r + Tr; tr++)                  → Loop reordering
     for (tc = c; tc < c + Tc; tc++)
      for (i = 0; i < k; i++)
       for (j = 0; j < k; j++)
        for (to = m; to < m + Tm; to++)              → Loop unrolling
         for (ti = n; ti < n + Tn; ti++)
          Ouput[to][tr][tc] += weight[to][ti][i][j]
            * Input[ti][S * tr + i][S * tc + j];
   }
   Store(Output);}
```
Figure 9: Pseudo Code of CONV Layer.

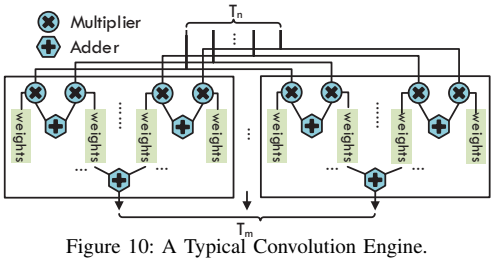

Figure 10: A Typical Convolution Engine.

2) Characterization of Current Design for In-situ AI:

a) Inference Task: The inference task is sensitive to the response time, especially for real-time applications. For instance, the response time should be less than 33ms for a real-time surveillance camera with 30FPS. For deep learning based tasks, batch processing is a common method to improve throughput. In Fig. 11, the latency of inference task (using AlexNet) increases along with the batch size on both mobile GPU and FPGA. To reduce latency, the inference task usually prefers to choose a small batch size. However, from the energy-efficiency perspective (i.e. images per second per watt), it is less desirable to utilize small batch size due to the low efficiency of FCN layers, which results from low reuse of FCN weights and the limited bandwidth of embedded platforms. Fig. 12 illustrates the time breakdown of inference task across various batch sizes. The FCN layers account for up to 50% of overall runtime with small batch sizes (i.e. 1, 2, and 4) on both FPGA and GPU.

For the power-sensitive IoT node, it is important to find an optimal batch size (B_{size}) where the inference task consumes the least energy under given runtime constraint. Considering the energy-efficiency, the optimal one should be the maximum batch size where latency is less than the end-user requirement. Therefore, it is necessary for us to design an analytical time model to guide the selection of the optimal batch size for the inference task.

b) Diagnosis Task: To minimize power consumption, our In-situ tasks are running on the same In-situ platform. Considering the requirement for inference task (whether it should be available 24/7 or not), there are two working modes for diagnosis task to concurrently run with inference task. One is Single-running mode, where the inference task is not always running. For example, the inference task runs

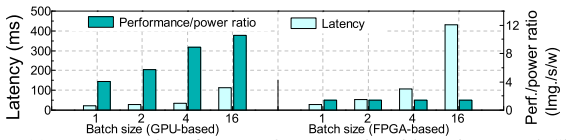

Figure 11: Latency and Performance/Power Ratio with Various Batch Sizes.

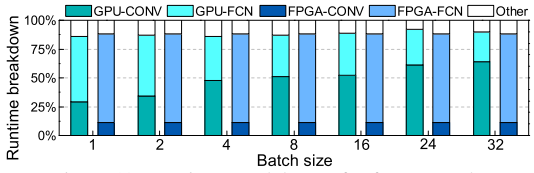

Figure 12: Runtime Breakdown of Inference Task.

in the daytime, while the diagnosis task works at night. In this mode, diagnosis task can share the same in-situ platform with inference task at different time slots. The other working mode is Co-running mode, where the inference task should be available 24/7. In this mode, diagnosis task has to run with inference task simultaneously on the in-situ platform. In this subsection, we demonstrate the characterization results when diagnosis task is working in the above modes.

Single-running mode: The diagnosis task is not sensitive to the response time. Thus, batching method can be utilized to improve the efficiency of FCN layers. On GPU, batching turns the computational pattern of FCN layers from matrix-vector multiplication to matrix-matrix multiplication. Thus, GPU's energy-efficiency (Performance to power ratio) improves with the batch size in Fig. 11. For FPGA, its energy-efficiency is fixed with various batch sizes. This is because its implementation does not consider the optimization of batch size in Fig. 9. We propose a batching optimization by introducing an extra batch loop in Fig. 13 (marked green). In this way, FCN weights can be reused for different samples in an input batch, which will greatly reduce data access of weights. As shown in Fig. 14, our batching optimization can also improve the energy-efficiency of FCN layers.

However, the situations are different for the CONV layers in Fig. 14. Batching can improve GPU energy-efficiency of CONV layers, while it yields no effect on the energy-efficiency of CONV layers running on FPGA. Fig. 15 compares the resource utilization of CONV layers between the two implementations. The resource utilization of GPU is determined by:

$$Util_{GPU} = \frac{Effective\ Computation}{Comp.\ Capacity \times Cycles} = \frac{Grid_{size}}{maxBlocks \times \left\lceil \frac{Grid_{size}}{maxBlocks} \right\rceil}. \quad (3)$$

Similarly, the resource utilization of FPGA can be expressed as:

$$Util_{FPGA} = \frac{N \times M}{T_n \times T_m \times \lceil N/T_n \rceil \times \lceil M/T_m \rceil}. \quad (4)$$

Batching method increases the dimension of data matrix D_m from $NK^2 \times RC$ to $NK^2 \times RC \times B_{size}$, which further increases $Grid_{size}$ and GPU utilization ($Util_{GPU}$). Therefore, batching method can improve the energy-efficiency of CONV layers on GPU. However, as shown in (4), the resource utilization of FPGA is not related with the batch size.

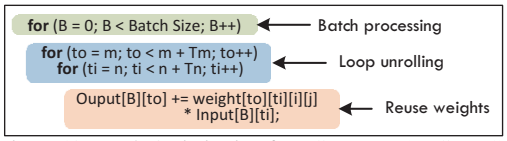

Figure 13: Batch Optimization for FCN Layer (R=C=K=1).

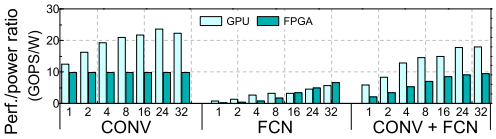

Figure 14: Perf./Power Ratio with Various Batch Sizes.

Therefore, as shown in Fig. 14, FPGA's energy-efficiency on CONV layers is fixed across various batch sizes and it is worse than that of GPU. What is more, the overall energy-efficiency (CONV+FCN) of GPU is better than that of FPGA.

Considering energy-efficiency is a common optimization goal for inference and diagnosis tasks, we choose GPU as our in-situ platform to deploy our In-situ AI tasks in Single-running mode. For the time-sensitive inference task, we should develop a time model to identify the optimal batch size so that it achieves the maximum performance/power ratio under the time requirement. For the diagnosis task without time requirement, big batch size will improve its energy-efficiency. However, processing with big batch size will easily run out of GPU memory. Therefore, we should propose a resource model to guide the maximum batch size selection and avoid the out-of-memory issue.

Co-running mode: In this working mode, the inference and diagnosis tasks will be running simultaneously. However, the interference between these two tasks is severe on GPU in Fig. 16. Our experiment result shows the latency of inference task will increase up to 3X due to the interference from the diagnosis task. For FPGA, its hardware-based design can effectively reduce the interference by separating hardware resource between these two tasks. Therefore, we take FPGA in Co-running mode. First, we should also ensure that the runtime of inference task meets the requirement of end-user with a time model. Second, we should leverage the opportunity of weight sharing to optimize the energy-efficiency of diagnosis task.

B. In-situ AI Architecture Design

In this subsection, we introduce our In-situ AI architecture design. Our design consists of two cases, namely Single-running mode design and Co-running mode design. As discussed in Section IV-A, GPU is responsible for the processing in Single-running mode, while the tasks in Co-running mode are accelerated by our FPGA-based design.

1) Single-running Mode: Note that although both inference and diagnosis tasks are deployed on GPU, their configurations (i.e. batch size) are determined by different models. For instance, for inference task, where latency is an important metric, the maximum batch size is decided by the time constraint derived from our time model. To avoid

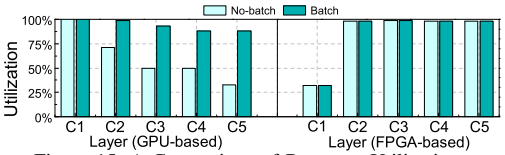

Figure 15: A Comparison of Resource Utilization.

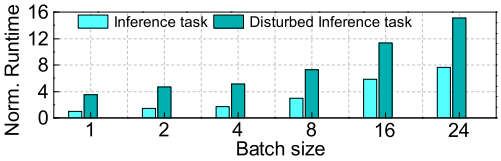

Figure 16: Interference between Inference and Diagnosis.

the out-of-memory issue, the maximum batch size of the diagnosis task is limited by the memory resource model. Next, we will discuss the time and resource models.

a) Time Model: To craft a time model for In-situ AI tasks, we need to model its runtime of CONV layers and FCN layers respectively. Since the CONV layers are always running on GPU, their performance on GPU (i.e. T_{CONV_GPU}) can be estimated as [28]:

$$T_{CONV\_GPU} = \frac{CONV_{ops}}{Achi.\ Perf.} = \frac{CONV_{ops}}{maxOPS \times Util}. \quad (5)$$

where $maxOPS$ is the maximum operations delivered by GPU per second and $Util$ is the resource utilization.

As discussed in Section IV-A2, the operation of FCN layers on GPU is changed from matrix-vector multiplication to matrix-matrix multiplication, which is similar to the computational pattern in CONV layers. Therefore, we can also use the time model of CONV layers to calculate the runtime of FCN layers on GPU. However, most of FCN layers are memory-intensive, and their performance is not only determined by GPU computational roof (i.e. $maxOPS$), but also limited by its memory bandwidth (i.e., MBW) [29].

$$Achieved\ Perf. = Min(maxOPS, CTM \times MBW), \quad (6)$$

where CTM is computational operations per memory access. For GPU based FCN layers, its $maxOPS_{GPU}$ and CTM_{GPU} are determined by (7) and (8), respectively:

$$maxOPS_{GPU} = 2Freq \times nCUDACore \times Util_{GPU}, \quad (7)$$

$$CTM_{GPU} = \frac{Layer_{ops}}{Data\ Access} = \frac{2M \times NK^2 \times RC \times B_{size}}{D_{in} + D_w + D_{out}}. \quad (8)$$

As shown in Fig. 8, D_{in} is $NK^2 \times RC \times B_{size}$, D_w is $M \times NK^2$ and D_{out} is $M \times RC \times B_{size}$. Note that K, R and C are 1 for the FCN layers.

b) Resource Model: For the diagnosis task, its configuration (i.e. B_{size}) is mainly limited by memory resource:

$$D_{in} + D_w + D_{out} \leq RAM_{capacity}. \quad (9)$$

2) Co-running Mode: As discussed in Section IV-A2, we leverage FPGA as our in-situ platform to perform inference and diagnosis tasks concurrently. To avoid the interference between inference and diagnosis tasks, we allocate dedicated computing resource to each task. The simplest solution is to duplicate the convolution engine in Fig. 10 and assign dedicated convolution engine for each task. As shown in Fig. 17, since diagnosis task has 9 independent inputs (9

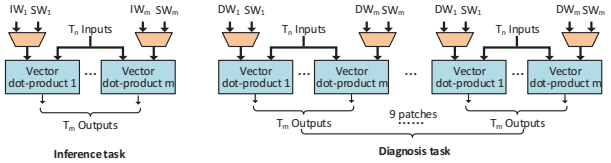

Figure 17: Weight Shared Architecture (WS) for In-situ AI tasks (IW: Inference Weight, SW: Shared Weight, DW: Diagnosis Weight).

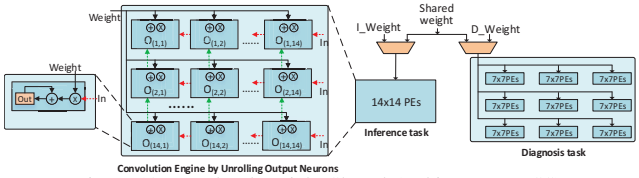

Figure 18: Two-level Weight Shared Architecture (WSS).

patches), it has 9 convolution engines and each is responsible for one patch input. To utilize the opportunity of weight sharing, we connect two weight sources, dedicated and shared weights, to each convolution engine. When layers are shared, the convolution engine will take the shared weight as its kernel weight; otherwise, choose its dedicated weight.

Although the input of diagnosis task is a partition of that of inference task, these two inputs have the same number of channels (3 in Fig. 4). Considering inference and diagnosis tasks have the same number of filters in each layer, they will have the same number of input feature maps (M) and output feature maps (N) in each layer. Therefore, we use the same unrolling parameters (T_m and T_n) to implement the convolution engine in Fig. 17. All the convolution engines have the same number (T_m) of vector dot-product units (each of them has T_n multiply-add unit). Although this architecture can utilize weight sharing between inference and diagnosis tasks (we name it as WS) to greatly reduce off-chip weight access, it has a problem of resource idleness due to its uniform unrolling strategy. Specifically, inference task has a larger output than that of diagnosis task in each layer (e.g. 55×55 vs 27×27 in the first layer), which further results in more computations in inference task. However, the uniform unrolling strategy allocates the same computing resource for each task. Therefore, there will be idle resource in diagnosis task due to its fewer computations. In our experiment, the convolution engines in diagnosis task will be idle during 75% of cycles.

To avoid the idleness, we should allocate the computing resource based on its computational load. As shown in (1), the computational load in each layer is proportional to its output feature map size ($R \times C$). To ensure inference and diagnosis tasks have the same runtime, their computing resource should be proportional to their output feature map sizes. Thus, the optimal unrolling strategy should be performed on its output neurons. In the left of Fig. 18, this strategy is illustrated by unrolling $T_r \times T_c$ output neurons (14×14). Each output neuron resides in a PE (Processing Element). At each clock cycle, T_r or T_c input neurons are received and shifted from right to left (red arrow) or down

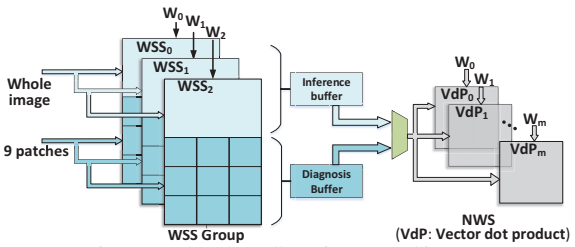

Figure 19: Our Overall In-situ AI Architecture.

to up (green arrow) across PEs and one kernel weight is broadcast to all PEs. Each PE keeps performing calculation for its corresponding output neuron by multiplying the input neuron with kernel weight and accumulating the partial result, till all the computations for this output neuron are accomplished. It takes $K \times K$ clock cycles to complete $T_r \times T_c$ output neurons. For each input, since the computational load in inference task is four times of that of diagnosis task, the computing resource for inference task should be four times as that of diagnosis task as well. In this way, inference and diagnosis tasks finish within the same time and the idleness is avoided.

In the right of Fig. 18, we use the convolution engine (unrolling output neurons) to design a weight shared architecture for the processing of CONV layers. The inference task is executed by one convolution engine with 14×14 PEs and the diagnosis task runs on 9 convolution engines with 7×7 PEs. We also allocate two sources, dedicated weight, and shared weight, to each convolution engine to enable the weight sharing in the first layers. Note that there are two levels of weight sharing in Fig. 18. One is weight sharing among different convolution engines and the other is the weight sharing in one convolution engine (one weight is shared by all PEs in one convolution engine). Therefore, we name our in-situ architecture in Fig. 18 as Weight-Share-Share (WSS).

Different from CONV layers, where the output neurons in the same output feature map share a kernel, each output neuron in FCN layers has its own kernel. Therefore, considering that the PEs in our convolution engine have the same kernel weight, our WSS is not applicable to FCN layers. We still need to take the convolution engine in Fig. 10 (No-Weight-Sharing, i.e. NWS) to process FCN layers. Therefore, as shown in Fig. 19, our overall In-situ AI architecture consists of two parts: WSS Group and NWS. WSS Group is responsible for CONV layers, while FCN layers run on NWS. To maximize the processing throughput, several Weight-Share-Share architectures (WSS) are combined to construct a WSS Group. In the WSS Group, the light blue area is for inference task while the dark blue area is for diagnosis task. Although all the WSS in WSS Group share the same input (whole image or 9 patches), they have their own kernel weights, so they can generate multiple output feature maps in parallel. The outputs are stored in inference and diagnosis buffers respectively. Finally, NWS chooses one input from the two

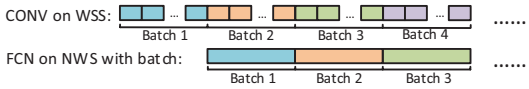

Figure 20: Pipeline-based FPGA Implementation.

buffers to perform the processing of FCN layers.

Moreover, WSS and NWS compose a pipeline (shown in Fig. 20) so that they can work in parallel. We also take the batch processing in FCN layers to improve the performance/power ratio on FPGA. To maximize the throughput of the pipeline-based design, the elapsed time on each pipeline stage should be identical. Since the batch optimization is only performed on FCN layers, the time consumption of FCN layers should be B_{size} times that of CONV layers, shown in Fig. 20. What is more, the sum of DSP slices used by WSS and NWS cannot exceed the total DSP resource of FPGA (i.e. Computing Resource Constraint in (10)).

$$WSS\ Group_{size} \times DSP_{WSS} + DSP_{NWS} \leq DSP_{total}. \tag{10}$$

Since WSS unrolls $T_r \times T_c$ output neurons, it consumes $T_r \times T_c$ (DSP_{WSS}) DSP slices. Similarly, DSP_{NWS} is $T_m \times T_n$ because NWS unrolls T_n input feature maps and T_m output feature maps. Based on the total operations in convolutional layer (1) and the number of DSP in WSS, we can derive the time spent in each layer:

$$T_{CONV\_FPGA} = \frac{\lceil M/WSS\ Group_{size} \rceil \times NK^2 \times \lceil R/T_r \rceil \times \lceil C/T_c \rceil}{Frequency}. \tag{11}$$

As discussed in Section IV-B1, the time spent on FCN layers is not only determined by FPGAs computational roof, but also limited by its memory bandwidth (MBW). Thus, the real runtime (T_{FCN_FPGA}) is:

$$T_{FCN\_FPGA} = Max(T_{FCN\_Comp.}, T_{FCN\_Mem.}), \tag{12}$$

where $T_{FCN\_Comp.}$ is $\frac{\lceil N/T_n \rceil \times \lceil M/T_m \rceil \times B_{size}}{Frequency}$ and $T_{FCN\_Mem.}$ is $Num_{data\ access}/MBW$. Thus, our overall runtime is:

$$T = 2Max(T_{All\_CONV\_FPGA} \times B_{size}, T_{All\_FCN\_FPGA}), \tag{13}$$

where $T_{All\_CONV\_FPGA}$ is the sum of time in each CONV layer (T_{CONV_FPGA}) and $T_{All\_FCN\_FPGA}$ is the sum of time in each FCN layer (T_{FCN_FPGA}). For the In-situ AI task, its optimization goal is to obtain the maximum processing throughput under the time requirement of end-users. Thus, we can find the optimal batch size (the maximum batch size that meets the time constraint in (14)) to implement our In-situ AI architecture for Co-running mode.

$$T \leq T_{user\_requirement}. \tag{14}$$

V. Evaluation

In this section, we first evaluate our In-situ AI node from microarchitecture design perspective. We mainly discuss how our analytical time models guide the optimal configuration in Single-running mode and compare our two-level shared architecture (WSS) with other traditional architectures (NWS and WS) in Co-running mode. Next, we demonstrate the overall benefits of our In-situ AI Cloud by comparing it with several alternatives. We use several metrics, including energy consumption, model update time

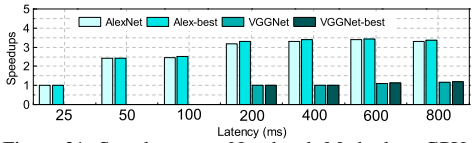

Figure 21: Speedups over Non-batch Method on GPU.

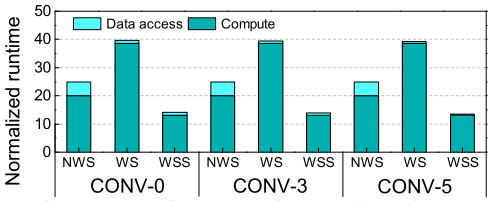

Figure 22: Runtime Comparison on CONV layers.

and data movement, in our evaluation. Our IoT platforms are Mobile GPU (i.e. NVIDIA TX1) and FPGA (i.e. Xilinx Virtex-7 VX690T).

A. In-situ AI Node

1) Single-running Mode: As discussed in Section IV-A2, the best architecture is GPU in this working mode. For the inference task on GPU, we develop a time model to achieve the maximum performance/power ratio under any given time requirement. Without the time model, it is common to use the non-batching method to realize the fast response time. We compare the implementation based on our time model with simple non-batching methods. We also compare our technique with the best case, which is obtained from brute-force based profiling, to understand the headroom for further improvement. Fig. 21 shows that our method yields an average of 3X speedup over the non-batching method and the speedup is especially high for the AlexNet based inference task. Note that compared to the resource underutilization on AlexNet based inference task, most of the computing resource is utilized by deeper networks such as VGGNet. Therefore, VGGNet based inference task only achieves an average of 1.1X speedup. In addition, as Fig. 21 shows, the performance achieved by our method close to the best case on both AlexNet-based and VGGNet-based inference tasks, indicating the effectiveness of our time model.

2) Co-running Mode: To evaluate our two-level shared architecture (WSS), we perform all CONV layers on NWS, WS and WSS and compare the runtime of these three architectures in Fig. 22. Note that all the architectures are implemented with the same number of PEs (2628). In this experiment, the weights of each layer are loaded from off-chip first and then the computation is performed. Besides, we study the impact of weight sharing by performing the convolutional layers under the following sharing strategies, CONV-0, CONV-3 and CONV-5, representing weights sharing of the first 0, 3 and 5 layers respectively. No matter how the sharing strategy is, our WSS outperforms the other two architectures in terms of compute time, while WS has the worst compute performance because of idleness in its convolution engines. Due to the optimization in weight

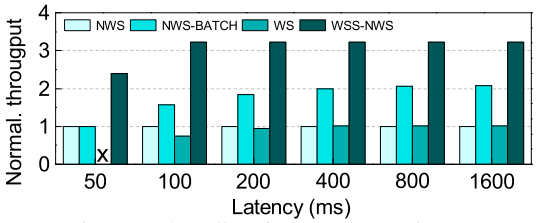

Figure 23: Overall Performance Comparison .

sharing, the time spent on data access in WSS is much less than that in NWS and decreases as the number of shared layers increases.

Next, we evaluate the overall performance of our pipeline-based In-situ AI architecture (WSS-NWS). Fig. 23 illustrates the maximum overall processing throughput for each architecture under various requirements of latency. For the processing of FCN layers, WS and WSS-NWS allow inputs in the same batch to share kernel weights to reduce off-chip access, while for NWS, two cases (NWS and NWS-batch) are applied (i.e. whether to share weights in a batch). In Fig. 23, due to the lack of batching optimization, NWS could not increase its processing throughput even under a loose requirement of latency, such as 800ms. For all the other architectures, an increase in throughput can be observed. However, this improvement of throughput ends when the performance of FCN layers is limited by the compute resource instead of off-chip bandwidth. Due to the underutilization of resource, WS is unable to meet the 50ms requirement (marked as x) and always produces the lowest throughput. Although NWS-batch performs better than NWS and WS, its maximum throughput (at 800ms) is still lower than our WSS-NWS's throughput under the most strict latency requirement (50ms). Among all the requirements of latency, our WSS-NWS can achieve the best processing throughput.

B. In-situ AI Cloud

In this section, we compare our In-situ AI (depicted in Fig. 24(d)) with the following three IoT systems. Fig. 24(a) illustrates the traditional IoT system equipped with unsupervised pre-training, where all the data acquired in IoT node is sent back to the Cloud and used to perform the pre-training. In Fig. 24(b), a diagnosis network is deployed into the Cloud, so that pre-training only focuses on the valuable data. Fig. 24(c) deploys the diagnosis task into the in-situ node to reduce data movement. Compared with Fig. 24(c), our In-situ AI in Fig. 24(d) leverages the weight sharing to further reduce the overhead of the transfer learning. All data in the Cloud is trained with NVIDIA Titan X. We mimic a real in-situ scenario, where IoT data is acquired incrementally. And our model is also incrementally updated based on the incremental data. First, we collect 100K images to train an initial model. Then our model is continually updated using 200k, 400k, 800k and 1200k images. We first compare data movement among these IoT systems during

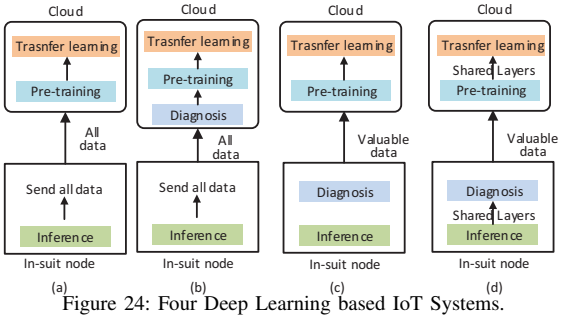

Figure 24: Four Deep Learning based IoT Systems.

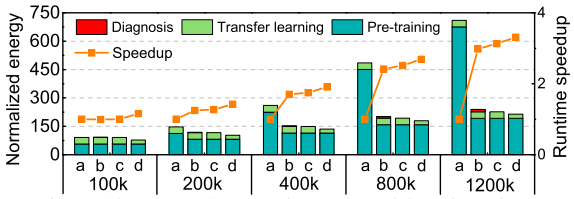

Figure 25: Energy Consumption and Model Update Time.

each update stage in TABLE II. Then, we characterize their energy consumptions and model update time in Fig. 25.

Comparing row 'a/b' and 'c/d' in TABLE II, we can see the amount of data movement reduced by our in-situ diagnosis task. Note that the reduced volume increases with the acquired data in IoT node. Initially, all the 100k data needs to be transferred to the Cloud to train the initial model. Then, as more and more data received, the accuracy of model can be gradually improved, which further leads to more and more data being correctly recognized in IoT node. Therefore, the unrecognized data that needs to be transferred become fewer and fewer in TABLE II.

Fig. 25 illustrates the energy consumptions in the Cloud and model update time among these four IoT systems. Our In-situ AI consumes the least energy due to the following reasons: (1) The re-training data is reduced by the diagnosis task (i.e. the difference between 'a' and 'b' in Fig. 25); and (2) The transfer learning is only performed on the last two CONV layers due to the weight sharing (i.e. the difference between 'c' and 'd' in Fig. 25). We also demonstrate the speedup on the model update time. Compared with the traditional IoT system (i.e. 'a' in Fig. 24), when the number of acquired data is 100k, our In-situ AI has a 1.15X speedup. Note that as the number of data increases, our In-situ AI will achieve more speedup on model update time (up to 3.3X).

VI. Related Work

GPU-based deep learning: Sirius [30] and DjiNN [31] bring the community the characterization of GPU acceleration server system running DNN services and provide insights into designing future warehouse-scale computer architectures for DNN services, while our work focuses on the characterization of DNN based applications on IoT nodes, especially for mobile GPU- and FPGA- based platforms. D^3NN [28] proposes an analytical time model for GPU accelerated CNN processing and its time model mainly focuses on the CONV layers. We further improve this model to include the modeling of FCN layers. P-CNN [32] presents a user satisfaction-aware CNN inference framework across different GPU architectures. To achieve high energy efficiency on mobile devices, Lane et al. [7] develop DeepX, a software accelerator for deep learning inference tasks. Compared with the above works, our In-situ AI includes more functions, such as unsupervised pre-training and autonomous data diagnosis.

TABLE II: A COMPARISON OF NORMALIZED DATA MOVEMENT

IoT system	100k	200k	400k	800k	1200k
a/b	1	1	1	1	1
c/d	1	0.72	0.51	0.35	0.29

FPGA implementation for deep learning: Zhang et al. [29] propose an analytical design scheme based on the roofline model to find the fastest FPGA implementation for CONV layers. They further extend their work to FCN layers to implement the entire CNN on FPGA [25]. However, their work overlooks the optimization of weight sharing and is not applicable to our In-situ AI tasks. Instead of a single large processor, [33] tries to improve the computational efficiency by designing multiple smaller specialized processors for different layers. However, their work mainly focuses on the CONV layers and does not consider the optimization on FCN layers. Zhu et al. [34] propose an OpenCL-based method to implement CNN on FPGA and they also compare the performance/power ratio per layer to a high-end GPU based implementation. However, they do not provide an analytical model to guide the best configuration selection for FPGA and GPU.

IoT system: To reduce the latency, there is a trend to perform the data analysis on IoT nodes. Li et al. [35] develop a sustainable in-situ server system to pre-process data, i.e., bringing computation to where the data is located. However, they do not consider deep learning based application as their workloads and include accelerators (e.g. FPGA and GPU) in their in-situ platform. Lane et al. [6] conduct an early resource characterization of deep learning on IoT devices. However, they overlook the problem of accuracy and always utilize a static model to perform the recognition.

VII. Conclusion

This work presents In-situ AI, the first autonomous and incremental computing framework and architecture for deep learning based IoT applications. We first characterize two In-situ AI tasks on two popular IoT devices and explore the design space and tradeoffs. To meet various IoT scenarios, we propose two working modes: Co-running and Single-running modes and craft analytical models for these two modes to guide their best configuration. We also develop a two-level weight shared architecture for Co-running mode. Compared with traditional IoT systems, In-situ AI can reduce data movement by 28-71%, which further yields 1.4X-3.3X speedup on model update and contributes to 30-70% energy saving.

ACKNOWLEDGMENT

This work is supported in part by NSF grants 1527535, 1423090, 1320100,1117261, 0937869, 0916384, 0845721 (CAREER), 0834288, 0811611, 0720476, by SRC grants 2008-HJ-1798, 2007-RJ-1651G, by Microsoft Research Trustworthy Computing, Safe and Scalable Multicore Computing Awards, and by three IBM Faculty Awards.

REFERENCES

[1] "Internet of Things in 2016: 6 Stats Everyone Should Know," http://www.fool.com/investing/general/2016/01/18/internet-of-things-in-2016-6-stats-everyone-should.aspx.

[2] "Deep Learning For Smart Cities," https://prateekvjoshi.com/2016/07/05/deep-learning-for-smart-cities.

[3] "How AI is Making Self-Driving Cars Smarter," http://www.roboticstrends.com/article/how_ai_is_making_self_driving_cars_smarter.

[4] "The Future of Humanity's Food Supply Is in the Hands of AI," https://www.wired.com/2016/05/future-humanitys-food-supply-hands-ai.

[5] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *ISCA*, 2016.

[6] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices," in *Proceedings of International Workshop on Internet of Things Towards Applications*, 2015.

[7] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices," in *IPSN*, 2016.

[8] Cisco, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," 2015.

[9] D. Crankshaw, X. Wang, J. E. Gonzalez, and M. J. Franklin, "Scalable training and serving of personalized models," in *LearningSys*, 2015.

[10] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009.

[11] "Snapshot serengeti," https://www.snapshotserengeti.org/.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[13] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[15] M. Noroozi and P. Favaro, "Unsupervised Learning of Visual Represention by solving Jigsaw Puzzles," in *ECCV*, 2016.

[16] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. Efros, "Context Encoders: Feature Learning by Inpainting," in *CVPR*, 2016.

[17] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised Visual Representation Learning by Context Prediction," in *ICCV*, 2015.

[18] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How Transferable Are Features in Deep Neural Networks?" in *NIPS*, 2014.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[20] "NVIDIA Jetson TX1," http://www.nvidia.com/object/jetson-tx1-module.html.

[21] "Virtex-7 VX690T," https://www.xilinx.com/products/boards-and-kits/dk-v7-vc709-g.html.

[22] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," *CoRR*, vol. abs/1410.0759, 2014.

[23] "CS231n: Convolutional Neural Networks for Visual Recognition," http://cs231n.github.io/convolutional-networks/.

[24] V. Volkov and J. W. Demmel, "Benchmarking GPUs to tune dense linear algebra," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2008.

[25] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *ICCAD*, 2016.

[26] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *ASPLOS*, 2014.

[27] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *ISCA*, 2016.

[28] M. Song, Y. Hu, Y. Xu, C. Li, H. Chen, J. Yuan, and T. Li, "Bridging the Semantic Gaps of GPU Acceleration for Scale-out CNN-based Big Data Processing: Think Big, See Small," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT)*, 2016.

[29] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *FPGA*, 2015.

[30] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang, and J. Mars, "Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers," in *ASPLOS*, 2015.

[31] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers," in *ISCA*, 2015.

[32] M. Song, Y. Hu, H. Chen, and T. Li, "Towards Pervasive and User Satisfactory CNN across GPU Microarchitectures," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017.

[33] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning," in *ISCA*, 2017.

[34] M. Zhu, L. Liu, C. Wang, and Y. Xie, "CNNLab: a Novel Parallel Framework for Neural Networks using GPU and FPGA," *CoRR*, vol. abs/1606.06234, 2016.

[35] C. Li, Y. Hu, L. Liu, J. Gu, M. Song, X. Liang, J. Yuan, and T. Li, "Towards Sustainable In-situ Server Systems in the Big Data Era," in *ISCA*, 2015.