IoBT-OS: Optimizing the Sensing-to-Decision Pipeline for the Internet of Battlefield Things

Dongxin Liu, Tarek Abdelzaher*, Tianshi Wang, Yigong Hu, Jinyang Li, Shengzhong Liu,
Matthew Caesar, Deepti Kalasapura
Department of Computer Science, University of Illinois at Urbana Champaign
Urbana, IL 61801, USA
*zaher@illinois.edu

Joydeep Bhattacharyya, Nassy Srour,
Maggie Wigness
DEVCOM Army Research Laboratory
Adelphi, MD 20783, USA

Jae Kim, Guijun Wang, Greg Kimberly,
Denis Osipychev
The Boeing Company
Seattle, WA 98124, USA

Shouchao Yao
Department of Computer Science, George Mason University
Fairfax, VA 22030, USA

***Abstract*—Recent concepts in defense herald an increasing degree of automation of future military systems, with an emphasis on accelerating sensing-to-decision pipelines at the tactical edge, reducing their network communication footprint, and improving the inference quality of intelligent components in the loop. These requirements pose resource management challenges, calling for operating-system-like constructs that optimize the use of limited computational resources at the tactical edge. This paper describes these challenges and presents *IoBT-OS*, an operating system for the Internet of Battlefield Things that aims to optimize decision latency, improve decision accuracy, and reduce corresponding resource demands on computational and network components. A simple case-study with initial evaluation results is shown from a target tracking application scenario.**

***Index Terms*—Internet of Battlefield Things; Edge Intelligence; Sensing-to-Decision Loops.**

I. Introduction

Recent worldwide defense trends reflect an increasing investment in automating various battlefield functions [1]–[3]. Initial indicators suggest that related applications have already impacted the course of recent conflicts [4]. Computer communications and networks, paired with sensing and machine intelligence, are at the center of enabling technologies for these applications, making the military domain a potential key focus for the intelligent edge and Internet of Things (IoT) research. An important goal is to support performant and resilient sensor-to-decision loops at the tactical edge [5], [6]. For example, one might want to reduce latency in neutralizing threats in the battlefield. The Internet of Battlefield Things (IoBT) [7] is an operating environment for future cyber-physical battlefields, where physical and computational assets must collaborate to produce effects. Prior work articulated IoBT challenges in resilience [8], [9], distributed computing [10], Bayesian learning [11], uncertainty estimation [12] and intelligent data fusion [13]–[15], among other areas [16]–[18]. This paper focuses on optimizing the efficiency and efficacy of the sensor-to-decision pipeline and offers an architecture, called IoBT-OS, to accomplish the optimization goals.

This work was supported in part by ARL Cooperative Agreement W911NF-17-2-0196, NSF CNS 20-38817, and The Boeing Company.

The need for IoBT-OS arises from the increasing complexity of networked computational resources in future battlefields. In general computing contexts, streamlining application development and operation necessitates the introduction of operating systems to address common performance and resilience challenges. Similarly, in a modern battlefield, where mission success depends in large part on computational artifacts, a new operating-system-like construct is in order. Its goal is to ensure that the execution of sensor-to-decision loops (that involve multiple intelligent devices and systems) meet the challenges arising from spatial distribution, accelerated mission-tempo, transient resources, and the potential presence of adversarial activity. IoBT provides the underlying support for ensuring performance and resilience of the networked computational and sensing substrate for the envisioned cyber-physical decision loops at the tactical edge.

The rest of this paper is organized as follows. Section II describes a functional decomposition of the sensing-to-decision pipeline. Section III overviews the IoBT-OS architecture. Sections IV, V, VI, and VII detail the four key components of IoBT-OS, respectively. A simple experimental case-study and its evaluation are is presented in Section VIII. The paper concludes with Section IX.

II. Background: The Decision Loop

A central concept for organizing IoBT functions is the multi-domain operation effect loop, proposed in prior work [5].

As mentioned in [5], it breaks down the data processing workflow into stages; namely, (i) detect (targets or threats), (ii) identify, (iii) track, (iv) aggregate (information), (v) distribute (to stakeholders), (vi) decide, and (vii) actuate. The execution of the loop starts with sensors that measure different signal modalities. It continues through communication components, and computational elements that execute appropriate machine analytics on sensor data. These analytics furnish information for decision making, such as detected target types and trajectories. The decisions usually involve selection of appropriate means of actuation (called *effects*) whose purpose is often to neutralize the identified threats. The success of IoBT support in executing this loop is measured against three key goals:

- *Tactical edge efficiency:* Recognizing that time is a decisive factor in gaining advantage, it is desired to accelerate the intelligent sensor-to-decision loops (ideally without loss of inference quality) and push key functionality towards the edge (to reduce dependency on large cloud-like infrastructure support). These advances allow pushing intelligent data analytics closer to the point of sensor data collection, thereby significantly shortening decision chains, while offering intelligent mission-informed data filtering as early as possible (at the edge) to reduce load on the possibly contested tactical network.
- *Edge resilience:* Recognizing that IoBT executes in adversarial settings, where new attacks on machine intelligence are possible (to disrupt decision loops and foil intelligent automation), it is important to develop foundations of resilience, especially in contexts where neural-network-based solutions are used for implementing the edge analytics. Such foundations must offer improved resilience to adversarial inputs, enhance risk analysis, speed-up sensor attack detection, and bound worst-case neural network outcomes in the presence of adversarial activity.
- *Tailored intelligence at the point of need:* Novel capabilities are needed at the tactical edge to take better advantage of heterogeneity, exploit multimodal sensing, and compute uncertainty. Examples include opportunistic exploitation of commodity radios as sensors, neural networks for inference in the frequency domain, distribution of machine learning models across heterogeneous edge systems, and sensor fusion to enhance target classification accuracy and reduce false positives (e.g., due to decoys) while meeting latency constraints.

Below, we focus primarily on edge efficiency. Issues of resilience and novel tailored intelligence services are beyond the scope of this paper.

III. IoBT-OS

To support the above goals, the IoBT-OS architecture includes three types of modules: (i) an edge AI efficiency library, (ii) mission-informed real-time data management algorithms, (iii) digital twin support, and (iv) offline training support.

The *edge AI efficiency library* comprises modules that encapsulate different functions for target/threat detection, identification, and tracking, using neural network components that have been compressed and optimized to execute in resource-scarce environments, while offering a controllable low latency to the application.

The *mission-informed real-time data management algorithms* segment and prioritize data processing by the supported real-time edge AI components. A key challenge is to allocate more resources to the processing of more critical stimuli, which requires a combination of (i) data segmentation by some notion of data importance or urgency, and (ii) prioritized allocation of capacity to process more important/urgent data first. These algorithms should further maximize resource utilization.

Digital twin support features a set of value-added auxiliary capabilities implemented on a high-end computing platform thereby allowing the system to exploit additional resources when available (e.g., when a connection to the higher-end server can be established). A primary capability of the digital twin is to replicate key IoBT system state and environmental conditions for purposes of conducting various compute-intensive functions centrally, such as search for an optimal system configuration, global anomaly detection, root cause analysis, and model checking.

Offline training support features solutions that pre-train various models ahead of deployment. Pre-training includes execution-time profiling in order to develop accurate latency models of different system components, as well as neural network training, especially in regimes where accurate data labeling of these data is very time-consuming. A challenge is therefore to exploit mostly unlabeled data to train the neural networks involved in the sensor-to-decision loop.

IoBT-OS builds upon earlier frameworks by the authors that describe a vision of edge intelligence systems [19] and highlight challenges in applying machine intelligence to the IoT edge [20], [21].

IV. Edge AI Efficiency Library

The first concern in sensing-to-decision loops is *resource efficiency*. There is an increasing incentive to push the processing of these loops as close to the data as possible, meaning that computations will need to be performed at the network edge or sensing device. A key advantage of such local processing lies in a shortened decision loop that removes reliance on remote resources (as such resources might not always be reachable in a hostile environment). Otherwise, an easy way to render battlefield assets useless would be to cut their connection to the remote computing equipment they need. The reliance on local resources instead calls for significant improvements in computational efficiency of the algorithms that must be executed at the edge. Another advantage lies in improving battery longevity of edge devices.

Three key functions need to be accelerated. First, it is important to be able to execute machine inference (e.g., target detection and classification tasks) efficiently. Second, to enable well-informed decisions, it is important to correctly estimate confidence in machine-inference results. Finally, when all else fails and local resources are deemed insufficient, efficient

compression solutions are needed to move data elsewhere for processing by more computationally involved machine inference algorithms downstream. Such compression solutions should be lightweight (on the resource-limited sensor side). Compression could be lossy, but lossy compression should be optimized for downstream machine inference, as opposed to human inspection. That's to say, data features needed to retain adequate accuracy of the downstream machine inference algorithm (as opposed to a human) should be preserved. Other features can be compressed away.

Consistently with the above needs, we next describe algorithms for (i) neural network reduction, (ii) efficient neural network confidence estimation, and (iii) compressive data offloading, implemented as part of the IoBT-OS edge-AI library and used in the experiment described later in this paper.

A. Neural Network Compression with DeepIoT

DeepIoT [22] is a framework that compresses the structure of deep neural networks for sensing applications. The compression is achieved by deciding the minimum number of elements in each layer in a manner informed by the topology and a global view of parameter redundancies of the network. DeepIoT borrows the idea of dropout, a widely-used deep learning regularization method, and exploits the network parameters themselves to determine the optimal dropout probabilities. A compressor neural network takes the model parameters as input, learns the parameter redundancies, and generates the dropout probabilities accordingly. The compressor neural network is optimized jointly with the original neural network through a compressor-critic framework that tries to improve the performance of the original sensing application while producing better dropout probabilities that can generate a more efficient network. DeepIoT is a unified approach for compressing all commonly used deep learning structures for sensing applications, including fully-connected, convolutional, and recurrent neural networks, and their combinations. In contrast to other approaches that sparsify large dense parameter matrices, DeepIoT converts parameters into a set of smaller dense matrices. As a result, the compressed neural network does not require additional storage for element indices and can be executed directly with the existing deep learning libraries. DeepIoT compressed structures greatly reduce resource consumption without affecting quality of inference, making it possible to deploy a broad range of deep neural networks on resource-constrained embedded devices at the tactical edge.

B. Confidence Estimation

The next challenge is to quantify the reliability of deep learning models. How, in particular, can principled uncertainty estimation reliably represent the correctness of model predictions? When deep learning is utilized to serve IoBT applications that require quantitative reliability assurances, principled uncertainty estimation is crucial. RDeepSense [23] is a recent effort that offers uncertainty estimates with theoretically established error bounds for sensing applications. RDeepSense employs a customizable function based on a weighted sum of negative log-likelihood and mean square error as the loss function. By adjusting the weighted sum, the underestimating impact of mean square error and the overestimation effect of negative log-likelihood are balanced. It was demonstrated that RDeepSense generates well-calibrated uncertainty estimates. In terms of resource efficiency, because RDeepSense emits a distribution estimate rather than a point estimate at the output layer, it can estimate uncertainty in a single run. RDeepSense significantly reduces execution time and energy usage when compared to sampling-based and ensemble-based approaches that involve executing a model k times for k samples.

C. Compressive Offloading

Compressive offloading [24], [25] is a framework for efficient offloading of deep learning computations. It is designed specifically for the thin-client edge-computing scenario, where sensor-side devices are assumed to be too resource constrained to run complex neural networks and/or data compression schemes (for offloading purposes). It allows the client device to run only *part* of the deep neural network locally, and offloads the rest of the computation to an edge server by transmitting intermediate results through the network. Moreover it employs an asymmetric antoencoder-based encoder/decoder that is lighter on the client side.

The goal of compressive offloading is to reduce the end-to-end deep-learning task latency that consists of the client processing latency, network transmission latency, and the server processing latency. The asymmetric autoencoder is composed of two parts. The first part is a light-weight data encoder that runs on the thin client. It quickly shrinks the dimensions of the input data, decreasing the size of the data that need to be offloaded. Consequently, the network latency is reduced due to the smaller size of data transmission. The second part is a decoder running on the edge server. It takes more efforts to reconstruct the original signal from the compressed feature maps, so the decoder is relatively more heavy-weight. However, since the edge server has a higher computing power, the absolute execution time of the decoder part is acceptable. The design of putting less computation on the thin client but more computation on the more powerful edge server reduces the total end-to-end task latency. Compressive offloading follows the principles of compressive sensing theory to greatly increase the data compression ratio while maintaining accuracy.

The encoder and decoder are further trained with an eye on optimizing downstream analytics. By placing the accuracy of downstream analytics as part of the encoder-decoder training loss-function, the compression learns to preserve features needed for maintaining quality of results, while removing other less important features, thereby achieving a higher compression ratio compared to generic encoder-decoder frameworks that aim to faithfully reconstruct all input detail.

V. Mission-Informed Real-time Data Management Algorithms

The second concern in sensing-to-decision loops is to develop scheduling algorithms that reduce decision latency.

While efficiency measures discussed above also reduce latency, they do not inherently consider specific time-constraints of different concurrent data processing functions. To address this issue, resource management algorithms are needed that can segment complex input data into subsets that have different mission-informed importance and/or urgency levels, then schedule the processing of these subsets in accordance with their importance and/or urgency. For example, if the mission is concerned with detecting tanks and similar military targets, then only those subsets of the scene that are suspected to contain such targets are of importance. Other parts of the scene could be processed at a lower priority.

The above observation calls for two types of mechanisms. First, it is desired to efficiently identify subsets of input data frames that are likely to contain targets of interest. Second, it is desired to schedule data processing in a manner consistent with their importance and latency constraints. Note that, these two problems are orthogonal to the optimizations described in the previous section. Specifically, while the previous section described optimizations to the *computational components* that process the input data (e.g., neural networks and data compression algorithms), the modules described in this section determine (i) *which subset of data* the aforementioned components will process and (ii) *at what priority*. The following two sections review solutions to such data segmentation and prioritization problems.

A. Self-cueing and Data Prioritization

Consider a video stream where some frames might include targets of interest. Our first challenge is to quickly determine which parts of the frames to pass for inspection to downstream neural networks (or other processing components), such that more critical data are processed first. This is a "chicken-and-egg" problem because we need to find important regions in the scene *before this scene is processed by the neural network* and thus before some level of semantic understanding of the scene is achieved. A self-cueing approach [26] builds on the assumption that strong temporal correlations exist in sensing data streams. For example, the consecutive frames in a video stream are highly similar. Thus, a self-cueing approach will process a full input frame at a lower frequency (e.g., once every few seconds) to localize and recognize all important targets, then use the observed target locations together with appropriately inferred motion vectors to approximately guess the locations of these same targets in the intermediate frames. Only the partial regions thought to contain the targets are processed in the intermediate frames. The remaining regions are skipped to save resources. The approach saves time by reducing the area that needs to be inspected by a deep neural network, or (in the case of data offloading) it saves by reducing the bandwidth consumption due to the data that need to be shared. To determine where a target is in intermediate frames, we use lightweight optical flow computations [27] to map the last-detected object locations onto the new frame (with appropriate region expansion to account for uncertainty). The key idea is inspired by the encoding done by common video codecs, except that it directly incorporates application-level semantics (e.g., the purpose of the mission) into the selection of regions to focus on. For example, motion of clouds in the sky need not be tracked. The selected partial regions gives rise to a set of executable tasks; each task is to process one region by downstream analytics. These tasks are then prioritized and scheduled, as described below.

B. Real-time Scheduling

We formulate the processing of the partial input regions (identified above) by downstream computational components as a real-time scheduling problem and solve it in a manner that meets data urgency and criticality constraints, while maximizing resource utilization. Such a problem was originally posed in [28], but has seen several extensions since [26], [29], [30]. Three key factors are determined by the scheduler: (i) prioritization, (ii) resource allocation, and (iii) task batching.

Prioritization decides the order of task processing, where mission-critical regions get a higher priority for a shorter response time. Different heuristics were developed to decide on priority assignment. For example, when the sensing data stream comes with physical distance information, distance-based [28] or relative-velocity-based criticality [29] can be applied. Alternatively, to optimize the overall detection accuracy, objects with a higher uncertainty in their locations during the tracking could be prioritized [26] over others. The prioritization policy is mission-specific.

Resource allocation decides the amount of computation to be allocated to each task. This allocation depends on the chosen execution fidelity. We developed multiple approaches to trade off fidelity and resource demand. First, instead of using a single-exit DNN, we proposed an imprecise computation model for DNNs [22], where multiple exit points are included at different depths. Earlier exits save time by skipping the remaining DNN layers, but offer a lower fidelity. Alternatively, we can dynamically change the spatial resolution of the input data by resizing the frames (or frame regions) before feeding them into the computational components [30]. The ensuing execution latency is correlated with the input data dimensions. Lower-resolution images are faster to process but are lower in fidelity. Finally, we can control the processing frequency of regions containing the same target according to their priority [26]. Intuitively, regions with important targets could be processed more frequently, while other parts may be skipped in some frames to save resources.

Task batching means we can combine multiple tasks that share the same processing kernel on the GPU. Batching typically requires that the tasks perform the same computations on inputs of the same dimension (e.g., same-size images). Batching exploits the advantages of parallel processing on modern GPUs; while batch response time increases slightly with batch size, it remains significantly lower compared to the latency of processing the tasks sequentially. Since batching may entail putting tasks of different priority into the same batch, it should be balanced carefully against prioritization to

attain a good trade-off between latency of high-priority tasks and overall GPU utilization.

Depending on the selected scheduling objectives, different scheduling algorithms were developed to optimize in the above multidimensional trade-off space, such as dynamic programming [22], [28], simple (batched) greedy heuristics [29], [30], and proportional planning algorithms [26]. In summary, mission-informed real-time data management algorithms simultaneously improve the general processing efficiency and prioritize the mission-critical subsets of data.

VI. The Digital Twin

The algorithms described above offer basic support for efficient and timely execution of sensing-to-decision loops at the tactical edge. These functions should operate even when the system is mostly disconnected from centralized rich resources. However, we do not assume that the system will always be disconnected. Rather, it is desirable to opportunistically exploit periods of temporarily improved connectivity to connect with higher-end resources that can run more resource-intensive value-added functions. Such functions include optimizing global system configuration (e.g., re-configuring the system to better adapt to transpired resource loss), performing global anomaly detection (e.g., detecting unusual spatially-distributed patterns that no individual sensor can see alone), performing root-cause analysis (when anomalies are present), and executing model-checking algorithms to ascertain properties of the current configuration (e.g., that it will always meet a specified end-to-end latency constraint). Since these functions rely on mirroring relevant aspects of system state at the component that performs the analysis, we summarily call the component responsible for executing them a *digital twin*. Below, we describe the overall digital twin architecture, present the protocol used to synchronize the twin with original system state, and highlight value-added functions performed by the twin (such as optimization and model-checking). Overall, the twin acts as an opportunistic "maintenance" component whose goal is to improve system operation and troubleshoot unexpected problems, when opportunity permits.

A. Overall Twin Architecture

The essential elements of a digital twin are a virtual representation, a physical asset, and the transfer of data/information between the two. Given that we are operating in a resource-constrained environment we expand on existing digital twin architectures [31] with various optimizations for data synchronization. Our architecture consists of two main subsystems in line with the definition of a digital twin [32]:

Digital twin manager: The first subsystem is the digital twin manager responsible for constructing the model of the cloned system and synchronizing its real-time state. It is thus composed of a *digital twin configuration service* and a *data synchronization service*. The configuration service reads the system configuration and instantiates the relevant entities in the digital twin. A library of entities (and their models) is maintained that can be used as building blocks in modeling the IoBT system. They include models of hardware components, such as sensors and cameras, as well as the physical elements, such as communication links and sensing channels. We also represent the library of algorithms used in various stages of the execution pipeline using reduced-order models. The model of an entity encapsulates its intrinsic properties and functionality (expressed in any suitable format, such as a set of equations or finite state machines). The model also exposes methods to which relevant state information can be passed and evaluated to capture the entity's interaction with the physical environment. The synchronization service interfaces between the physical system and the digital twin model to synchronize relevant entity data and perform twin state updates.

Twin analytics: The second subsystem is the library of modules that run value-added analytics on the digital twin. These include modules for optimizing the system configuration, checking safety properties of the system, performing anomaly detection and doing root cause analysis. Below, we elaborate these subsystems.

B. System/Twin Synchronization

A core component of the architecture is the data synchronization service running on the Digital Twin manager that keeps the twin up-to date with the current system state. We define three classes of parameters that have to be synchronized in order to capture an accurate representation of the real-time system state: *(i) Physical environment parameters* that describe the current state of the external environment, such as sensor measurements and estimated noise. *(ii) Configurable parameters* that can be modified during initial setup or at run-time, such as chosen radio power settings, sensing (e.g., camera) resolution, and various algorithmic settings (e.g., averaging window for sensor data). Their values can be set by optimization functions on the digital twin. *(iii) System composition parameters* that represent the set of entities available in the system and their properties and interconnections. They can control flow topology, such as the sensing modalities used.

As shown in Figure 1, we define a Twin Communication Protocol which delineates the conditions for synchronization. Since we have limited bandwidth resources, it is key that the system only sends data that are important to the application. Synchronization decisions are made using the concept of intent-based transmissions. For instance, if the digital twin has been configured to generate a system configuration that minimizes the end-to-end latency, the implicit intent guiding synchronization would be changes to state variables that have a direct impact on the latency computation. Every entity in the system maintains a local copy its state last synchronized with the digital twin in the cloud. We refer to this as the *shadow twin*. The real-time measurements in the system entity are compared with the shadow twin state. Significant changes trigger synchronization with the digital twin.

C. Value-Added Functions

A library of analytical functions are leveraged to produce insights using the context generated by the digital twin. This li-

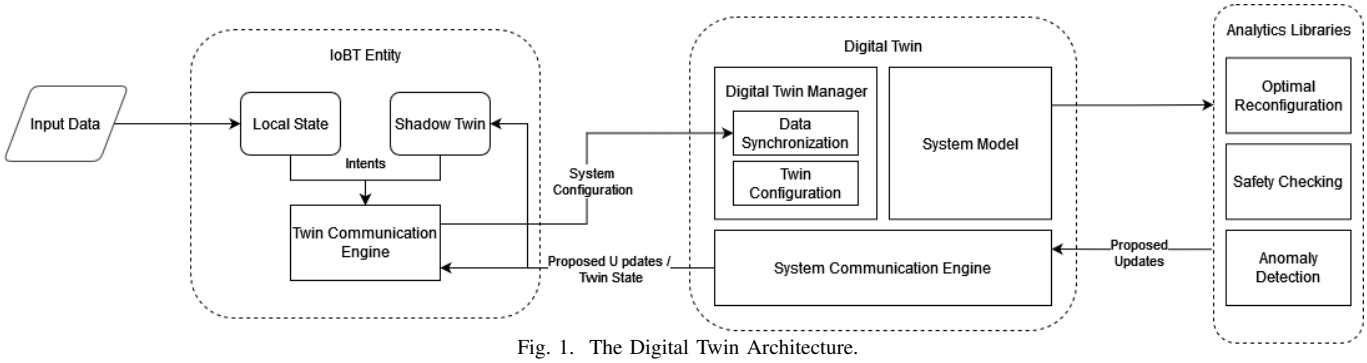

Fig. 1. The Digital Twin Architecture.

brary is extensible and newer modules can be added according to the digital twin's application. We describe three functions that provide useful insights for the IoBT system. (i) *Optimal (re-)configuration*: The initial system configuration describes the set of components and how they are connected. However, there is a huge re-configuration space associated with the hardware and software components in the system. An optimizer explores the state space and produces the best system configuration that can minimize or maximize a specified metric. (ii) *Safety-checking*: There are many parameters that impact the performance of the system. This module formally verifies that required safety and liveness constraints will hold true in the current system state. (iii) *Anomaly detection and root-cause analysis*: The digital twin models the normal operating state of the system. The anomaly detector analyzes synchronization data to identify and localize inconsistencies. A root cause analysis module locates the original problem. This information provides additional context to the optimizer module to produce a new configuration resistant to the anomaly.

VII. Offline Training Support

Finally, the efficacy of much of the online processing described earlier depends on how well the underlying components were *pre-trained* prior to deployment. In this paper, we focus on two aspects of such pre-training. First, accurate latency (and other resource consumption) models are needed for the computational elements involved in online processing. This is the responsibility of *offline profiling*. Since neural networks are most resource-intensive of all IoBT-OS edge components, we focus on profiling their execution. Second, training the neural networks requires a large amount of labeled data. However, operationally, unlabeled data are much easier to collect. Thus, we describe the use of contrastive learning to train neural networks using mostly unlabeled data.

A. Offline Profiling

Solutions such as DeepIoT [22], reduce the size of neural networks but do not inherently track the impact of such reduction on execution latency. Neural network compression (to fit on the tactical edge) is better cast as a problem of trading off inference quality versus execution latency on the target device. Such a problem formulation requires a latency model that describes the relationship between the model execution time and the neural network parameters that affect output quality. This section focuses on computing such a latency model. Three characteristics of neural networks make it very challenging to build a good latency model. First, the huge parameter space of modern neural networks makes it infeasible to determine execution time of all possible network configurations by brute force search. Second, the execution latency shows a non-monotonic and nonlinear relationship [33], [34] with key neural network parameters, such as the number of output channels in a convolutional layer. This makes it harder to extrapolate from limited configuration testing results. Third, neural networks often require relatively complex hardware and software stacks that hide several internal optimizations of their own, often resulting in significant unpredictable effects on latency [35].

Our approach to profiling, described in an earlier publication [35], utilizes knowledge of hardware and software to reduce the amount of required empirical testing, while using data-driven methods to reduce the complexity of required analytical modeling. For convolutional layers that take up the vast majority of the execution time of many neural network models, the solution identified two key non-linearities; one is attributed to a hardware quantization effect arising from optimizing block-level parallelism on GPUs; the other is traced to a software auto-tuning effect, respectively. By modeling and accounting for these nonlinearities, the approach accurately predicts execution latency, while reducing empirical profiling needs to manageable amounts.

B. Self-Supervised Contrastive Learning

Deep neural networks have demonstrated outstanding performance on sensing tasks [36], but require a large amount of labeled data in the training process. While it is possible to collect large amounts of sensory measurements, accurate labeling of these data is very time-consuming. To reduce the data labeling burden and efficiently exploit the unlabeled data, we apply self-supervised contrastive learning to train our offline neural network models.

The key idea of self-supervised contrastive learning is to first learn features (called representations) of input sensory measurements, then build the downstream task (such as classification or regression) models on the features instead of the original inputs. Thus, our offline training can be divided into two parts: 1) *self-supervised contrastive feature learning*, and 2) *downstream task training*.

Self-supervised contrastive feature learning aims at training an encoder that maps the input data into low-dimensional latent features in a self-supervised way. Only unlabeled data are needed in this step. To train the encoder, different views of each input data sample are generated through a data augmentation algorithm. The encoder would take the augmented views as input and generate the corresponding features. The features are then mapped, by a projection head, to a latent space where the contrastive loss is calculated. The augmented views from the same input are called positive pairs and augmented views from different inputs are negative pairs. The contrastive loss is then defined for the contrastive prediction task which aims at pulling together the positive pairs and at the same time pushing apart the negative pairs. After minimizing the contrastive loss, the encoder would be freezed. The low-dimensional features learnt in this step would extract intrinsic information from the raw input data and hence support the downstream machine learning tasks.

Downstream task training aims at training neural networks (e.g., a classifier) based on the features learnt by the self-supervised contrastive learning in a supervised way. The encoder takes the raw data as input and then generate the corresponding feature. This feature facilitates the downstream training. A relatively simple model (e.g., a linear classifier) usually work well. Thus, a small labeled dataset would be enough to train the downstream model with high performance.

Our work also takes frequency domain characteristics into consideration and designs encoder and data augmentation algorithms from a time-frequency perspective, demonstrating performance gains [37]–[39].

VIII. An Evaluation Case Study

To illustrate the use of the above components, we describe a simple application, where acoustic sensors, seismic sensors, and cameras were used to detect and classify targets, while measuring end-to-end latency and classification accuracy.

A. Hardware Set-up and Execution Pipeline

We use a vehicle detection and classification application to study the efficiency of IoBT-OS. In our setup, a Geophone and microphone are used to collect seismic and acoustic signals for preliminary vehicle detection and classification. The sensing devices consist of a Raspberry Shake (Raspberry Pi + one vertical-axis geophone) [40], a microphone array, and a portable power bank. A camera is also used to take pictures of the targets for the final visual confirmation on a separate edge server. We used a Jetson Nano as the server.

The execution pipeline is shown in Figure 2. The Geophone and microphone collect the seismic and acoustic signals from the environment. The seismic and acoustic data are then processed to detect and classify targets in real time. The target classifier is a neural network trained with our offline training techniques mentioned in Section VII. When the target classifier becomes confident enough in its prediction that a given target is found (i.e., the confidence level is larger than a threshold), a message is sent to a camera that takes a picture of the target for confirmation. In order to reduce the transmission delay, the picture is first compressed by our compressive offloading framework and then sent to the edge server. A vision-based object detector (based on YOLO [41]) on the server processes the picture and confirm the target (or not). We use an object detector that was compressed using our DeepIoT framework to reduce its inference latency and computational resource consumption.

B. Experimentation Results

To evaluate the performance of IoBT-OS, we deployed our devices on the grounds of the DEVCOM Army Research Laboratory Robotics Research Collaboration Campus (R2C2) [42] and collected seismic and acoustic signals, while different ground vehicles were driven around the site. Data of three different targets: a Polaris all-terrain vehicle, a Chevrolet Silverado, and Warthog UGV were collected. Each target repeatedly passed by the sensors. The total length of the experiment was 115 minutes, spread roughly equally across the three targets. The seismic data was collected at 100 Hz and the acoustic data was collected at 16000 Hz. To reduce overhead, the acoustic data was low-pass filtered at 400 Hz and high-pass filtered at 25 Hz, then down-sampled to 100 Hz. Based on the collected data, we study the decision accuracy and latency of IoBT-OS. A camera was employed to simultaneously record video of the target.

1) Decision Accuracy: We first studied the accuracy of target classification based on seismic and acoustic data. We applied the DeepSense neural network architecture [36] as the structure of the target classifier. Figure 3 illustrates its structure. Our previous work [43] has shown that physical sensing signals have sparser and more compact representations in the frequency domain. Thus, we used a spectrogram instead of time domain measurements as the input to the classifier. We trained the target classifier with the self-supervised contrastive learning framework mentioned in Section VII-B.

TABLE I
Decision Accuracy

Three-Layers CNN	Target Classifier	Target Classifier+SCL
73.6%	89.1%	91.2%

We used roughly two thirds of the data for training and one third for testing. The seismic and acoustic data were cut into segments of one second each using a sliding window without overlap. We built a three-layers convolutional neural network (CNN) as a baseline and also compared the accuracy of our target classifier with and without self-supervised contrastive learning (SCL). The results are shown in Table I. We observe that our target classifier has much higher recognition accuracy than the CNN baseline (89.1% vs 73.6%). And the that self-supervised contrastive learning further improves the accuracy from 89.1% to 91.2%.

In the experiment, the seismic and acoustic signals were continuously fed into the target classifier. Given N consecutive windows (i.e., N seconds worth of data) we took majority

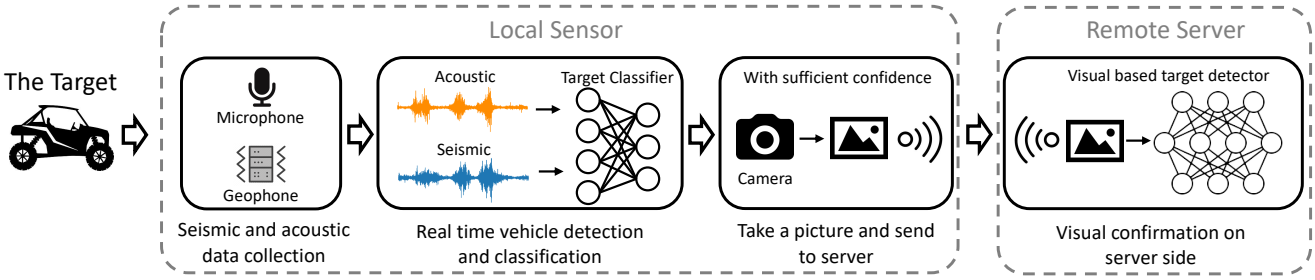

Fig. 2. The Execution Pipeline.

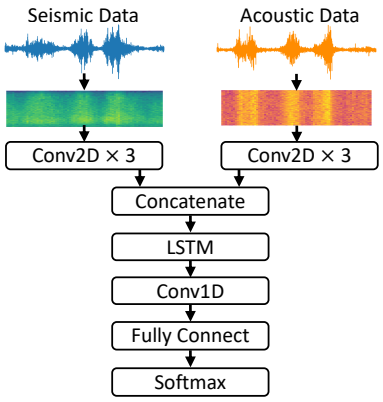

Fig. 3. The Target Classifier.

vote as the final classification result. We varied N from $1s$ to $20s$ and studied the classification accuracy for different data lengths to understand the trade-off between quality and latency. The results are shown in Figure 4. We observed that our target classifier trained with self-supervised contrastive learning (the blue curve) always performs the best, and shows a 99% classification accuracy given 20 seconds of seismic and acoustic data.

We also studied the effectiveness of the self-supervised contrastive learning framework at utilizing unlabeled data. Specifically, we assumed that 1%, 2%, 5%, 10%, 20%, 30%, 40%, and 50% of the training data has labels (and that the remaining training data is unlabeled). We then compared the performance of the target classifier trained with supervised learning (i.e., with the labeled data only) and the self-supervised contrastive learning (trained with both labeled and unlabeled data) given the above ratios of labeled data. Figure 5 demonstrates the results. It can be seen that the target classifier trained with our self-supervised contrstive learning (the blue curve) performs much better than that trained in a supervised manner (the orange curve). This illustrates the efficiency of our self-supervised contrastive learning framework in utilizing unlabeled data.

2) Confidence Estimation: A camera was used in the experiment as a third modality to confirm the target. While in reality, the camera was on all the time, we imagine that it was activated only when the target classifier based on acoustic and seismic sensors had sufficient confidence to ask the camera for confirmation. The idea was for these sensors to remain stealthy and refrain from communication unless sufficient confidence has accumulated that a target of the desired class in nearby. We applied rDeepSense [23] to do the confidence estimation. To evaluate the performance of rDeepSense, we divided all the prediction results in the testing set into 11 intervals according to their confidence scores: [0-0.1), [0.1,0.2), [0.2,0.3), ..., [0.9,1), 1, and then calculated the average confidence score as well as the average percentage of correct classification for each interval. Ideally, the two should be equal. The results are shown in Figure 6. We can observe that the difference between the average confidence score and average classification accuracy under each interval is very small. This demonstrates the effectiveness of our confidence estimation strategy.

TABLE II
INFERENCE TIME FOR THE TARGET CLASSIFIER

	Three-Layers CNN	**Target Classifier (SCL)**
Raspberry Pi 3B+	10.6 ms	66.0 ms
Raspberry Pi 4	4.7 ms	17.7 ms

3) Decision Latency: In this part, we study the performance of IoBT-OS on decision latency. We begin with the inference time of the target classifier based on acoustic and seismic sensors. The target classifier runs on the local sensors and continuously takes the seismic and acoustic data as input. It is required that the classifier's inference speed should be shorter than the interval at which seismic and acoustic data comes in. Otherwise, data will back up. Table II demonstrates the inference time (for a one second data segment as input) on different devices. We observe that the inference times of our target classifier are slightly larger than those of the three-layers CNN baseline. However, they are much smaller than one second. For example, on the Raspberry Pi 4, it takes only 17.7 ms to process the 1 second data window. This means that our classifier could generate its classification result in time.

Another component of latency in the decision loop comes when visual confirmation is needed. The first non-trivial part of that latency is the time needed for image transmission from camera to edge server. (Note that, the time needed to ask the camera to take a picture is comparatively small and so is neglected.) We thus study the transmission delay for the picture. In order to reduce the network delay, the picture is

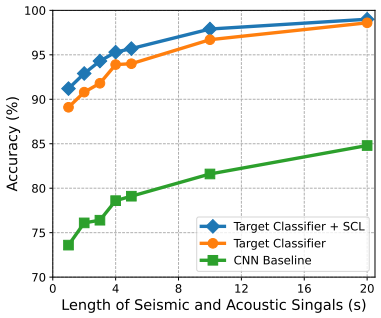

Fig. 4. Accuracy under different data lengths.

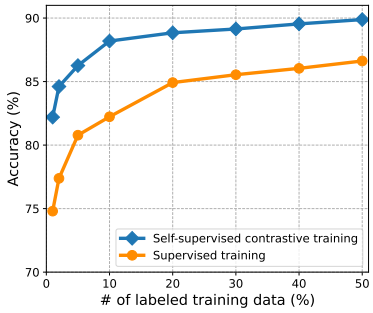

Fig. 5. Accuracy under different # of labels.

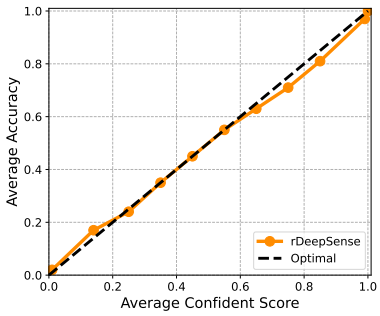

Fig. 6. Performance of confidence estimation.

TABLE III
PICTURE TRANSMISSION DELAY

	Encoding Time		**Size**	**Network Delay**			**Decoding Time**
	Raspberry Pi 3B+	Raspberry Pi 4		LoRa 12.5Kbps	LoRa 21.9Kbps	3G 0.1Mbps	Jetson Nano
Compressive offloading	44.9 ms	25.1 ms	0.08 MB	55.3 s	31.6 s	7.0 s	45.5 ms
JPEG	53.2 ms	26.3 ms	0.23 MB	153.2 s	87.4 s	19.1 s	86.5 ms
Raw	0 ms	0 ms	0.98 MB	677.4 s	387.1 s	85.7 s	0 ms

compressed before transmission with our compressive offloading framework. We compared the performance of compressive offloading with that of JPEG compression and with directly sending the raw picture. The results are shown in Table III. The overall processing delay for transmitting a picture consists of three parts: encoding time on the local sensor, network delay, and decoding time on the serve side. We observed that when network bandwidth is low (as might be the case in a battlefield network), the network delay contributes the most to the overall picture transmission delay. Our compressive offloading framework compressed the raw picture (0.98MB) to a much smaller size (0.08MB) compared with JPEG (0.23MB), and hence led to a much shorter network delay. Compared with network delay, the encoding and decoding time is much smaller (ms vs s), and compressive offloading has a smaller $encoding + decoding$ time compared with JPEG.

Finally, we study the inference time of the visual object recognition on the edge server side. We use YOLO as our visual object detector and apply DeepIoT to compress it in order to get a shorter inference time. We compare the inference time before and after applying DeepIoT on YOLO. The results are shown in Table IV. In Jetson Nano, DeepIoT approximately reduce the inference time of YOLO by 50%.

TABLE IV
INFERENCE TIME FOR THE VISUAL OBJECT DETECTOR

	YOLO	**YOLO + DeepIoT**
Jetson Nano	94.5 ms	48.2 ms

Compressive offloading and DeepIoT could reduce the network delay and the inference time of the visual confirmation, respectively. With much shorter latency, they would lead to comparable performance on visual confirmation. We fed 300 pictures compressed by compressive offloading to our YOLO compressed by DeepIoT, no prediction error was found.

The above concludes our evaluation. We do not report results on the digital twin because it is not involved in the run-time sensing-to-decision loop. In contrast, ablation study results, such as those reported above, can be used by the twin's optimization engine in future deployments of the pipeline to configure it in a manner that meets specified latency and quality trade-offs.

IX. CONCLUSIONS

The paper presented results of an experiment, where a collection of edge-AI acceleration techniques were jointly used to reduce the latency and improve the inference quality of a small sensing-to-decision loop. The solutions used in these paper are incorporated into the IoBT-OS architecture. An ablation study details the impact of different components on overall performance, showing that significant improvements are achieved.

REFERENCES

[1] A. B. Assessment and C. P. Brief, "US military investments in autonomy and AI," 2020.

[2] S. Petrella, C. Miller, and B. Cooper, "Russia's artificial intelligence strategy: the role of state-owned firms," *Orbis*, vol. 65, no. 1, pp. 75–100, 2021.

[3] M. C. Horowitz, "Artificial intelligence, international competition, and the balance of power," *2018*, vol. 22, 2018.

[4] J. F. Antal, *7 Seconds to Die: A Military Analysis of the Second Nagorno-Karabakh War and the Future of Warfighting*. Casemate, 2022.

[5] T. Abdelzaher, A. Taliaferro, P. Sullivan, and S. Russell, "The multi-domain operations effect loop: from future concepts to research challenges," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II*, vol. 11413. International Society for Optics and Photonics, 2020, p. 1141304.

[6] S. Russell, T. Abdelzaher, and N. Suri, "Multi-domain effects and the internet of battlefield things," in *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*. IEEE, 2019, pp. 724–730.

[7] S. Russell and T. Abdelzaher, "The internet of battlefield things: the next generation of command, control, communications and intelligence (c3i) decision-making," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 737–742.

[8] T. Abdelzaher, N. Ayanian, T. Basar, S. Diggavi, J. Diesner, D. Ganesan, R. Govindan, S. Jha, T. Lepoint, B. Marlin *et al.*, "Toward an internet of battlefield things: A resilience perspective," *Computer*, vol. 51, no. 11, pp. 24–36, 2018.

[9] S. Liu, S. Yao, Y. Huang, D. Liu, H. Shao, Y. Zhao, J. Li, T. Wang, R. Wang, C. Yang *et al.*, "Handling missing sensors in topology-aware iot applications with gated graph neural network," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 3, pp. 1–31, 2020.

[10] T. Abdelzaher, N. Ayanian, T. Basar, S. Diggavi, J. Diesner, D. Ganesan, R. Govindan, S. Jha, T. Lepoint, B. Marlin *et al.*, "Will distributed computing revolutionize peace? the emergence of battlefield IoT," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1129–1138.

[11] A. D. Cobb, B. A. Jalaian, N. D. Bastian, and S. Russell, "Robust decision-making in the internet of battlefield things using bayesian neural networks," in *2021 Winter Simulation Conference (WSC)*. IEEE, 2021, pp. 1–12.

[12] B. M. Marlin, T. Abdelzaher, G. Ciocarlie, A. D. Cobb, M. Dennison, B. Jalaian, L. Kaplan, T. Raber, A. Raglin, P. K. Sharma *et al.*, "On uncertainty and robustness in large-scale intelligent data fusion systems," in *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*. IEEE, 2020, pp. 82–91.

[13] E. Blasch, T. Pham, C.-Y. Chong, W. Koch, H. Leung, D. Braines, and T. Abdelzaher, "Machine learning/artificial intelligence for sensor data fusion–opportunities and challenges," *IEEE Aerospace and Electronic Systems Magazine*, vol. 36, no. 7, pp. 80–93, 2021.

[14] S. Liu, S. Yao, J. Li, D. Liu, T. Wang, H. Shao, and T. Abdelzaher, "Giobalfusion: A global attentional deep learning framework for multisensor information fusion," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–27, 2020.

[15] S. Yao, Y. Zhao, H. Shao, D. Liu, S. Liu, Y. Hao, A. Piao, S. Hu, S. Lu, and T. F. Abdelzaher, "Sadeepsense: Self-attention deep learning framework for heterogeneous on-device sensors in internet of things applications," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1243–1251.

[16] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–12.

[17] T. Li, J. Huang, E. Risinger, and D. Ganesan, "Low-latency speculative inference on distributed multi-modal data streams," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 67–80.

[18] D. Basu, D. Data, C. Karakus, and S. Diggavi, "Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[19] S. Yao, Y. Hao, Y. Zhao, A. Piao, H. Shao, D. Liu, S. Liu, S. Hu, D. Weerakoon, K. Jayarajah *et al.*, "Eugene: Towards deep intelligence as a service," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1630–1640.

[20] S. Yao, Y. Zhao, A. Zhang, S. Hu, H. Shao, C. Zhang, L. Su, and T. Abdelzaher, "Deep learning for the internet of things," *Computer*, vol. 51, no. 5, pp. 32–41, 2018.

[21] T. Abdelzaher, Y. Hao, K. Jayarajah, A. Misra, P. Skarin, S. Yao, D. Weerakoon, and K.-E. Årzén, "Five challenges in cloud-enabled intelligence and control," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 1, pp. 1–19, 2020.

[22] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2020.

[23] S. Yao, Y. Zhao, H. Shao, A. Zhang, C. Zhang, S. Li, and T. Abdelzaher, "Rdeepsense: Reliable deep mobile computing models with uncertainty estimations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, pp. 1–26, 2018.

[24] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.

[25] ——, "Deep compressive offloading: Speeding up edge offloading for ai services," *GetMobile: Mobile Computing and Communications*, vol. 25, no. 1, pp. 39–42, 2021.

[26] S. Liu, X. Fu, M. Wigness, P. David, S. Yao, L. Sha, and T. Abdelzaher, "Self-cueing real-time attention scheduling in criticality-aware visual machine perception," in *Proceedings of the 28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022.

[27] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool, "Fast optical flow using dense inverse search," in *European Conference on Computer Vision*. Springer, 2016, pp. 471–488.

[28] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *In Proc. IEEE Real-time Systems Symposium (RTSS)*, December 2020.

[29] S. Liu, S. Yao, X. Fu, H. Shao, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "Real-time task scheduling for machine perception in intelligent cyber-physical systems," *IEEE Transactions on Computers*, 2021.

[30] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, "On exploring image resizing for optimizing criticality-based machine perception," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021.

[31] Y. Qamsane, C.-Y. Chen, E. C. Balta, B.-C. Kao, S. Mohan, J. R. Moyne, D. M. Tilbury, and K. Barton, "A unified digital twin framework for real-time monitoring and evaluation of smart manufacturing systems," *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pp. 1394–1401, 2019.

[32] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in cps-based production systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 12 2017.

[33] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17. New York, NY, USA: ACM, 2017, pp. 4:1–4:14.

[34] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. F. Abdelzaher, "FastDeepIoT: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, SenSys 2018, November 4-7, 2018*. Shenzhen, China: ACM, 2018, pp. 278–291.

[35] J. Li, R. Ma, V. S. Mailthody, C. Samplawski, B. Marlin, S. Chen, S. Yao, and T. Abdelzaher, "Towards an accurate latency model for convolutional neural network layers on gpus," in *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, pp. 904–909.

[36] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 351–360.

[37] D. Liu, T. Wang, S. Liu, R. Wang, S. Yao, and T. Abdelzaher, "Contrastive self-supervised representation learning for sensing signals from the time-frequency perspective," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–10.

[38] D. Liu, P. Wang, T. Wang, and T. Abdelzaher, "Self-contrastive learning based semi-supervised radio modulation classification," in *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, pp. 777–782.

[39] D. Liu and T. Abdelzaher, "Semi-supervised contrastive learning for human activity recognition," in *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2021, pp. 45–53.

[40] "Raspberry shake," https://raspberryshake.org/, accessed: 2022-04-09.

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[42] "Pn13-68 (us army, apg-g-field and graces quarters-aerostat platforms)," https://www.nab.usace.army.mil/Missions/Regulatory/Public-Notices/Article/492714/pn13-68-us-army-apg-g-field-and-graces-quarters-aerostat-platforms-2013-61848-m/, accessed: 2022-04-09.

[43] S. Yao, A. Piao, W. Jiang, Y. Zhao, H. Shao, S. Liu, D. Liu, J. Li, T. Wang, S. Hu *et al.*, "Stfnets: Learning sensing signals from the time-frequency perspective with short-time fourier neural networks," in *The World Wide Web Conference*, 2019, pp. 2192–2202.