Meta-Learning Based Runtime Adaptation for Industrial Wireless Sensor-Actuator Networks

Xia Cheng, Mo Sha
Knight Foundation School of Computing and Information Sciences
Florida International University
{xcheng,msha}@fiu.edu

***Abstract*—IEEE 802.15.4-based industrial wireless sensor-actuator networks (WSANs) have been widely deployed to connect sensors, actuators, and controllers in industrial facilities. Configuring an industrial WSAN to meet the application-specified quality of service (QoS) requirements is a complex process, which involves theoretical computation, simulation, and field testing, among other tasks. Since industrial wireless networks become increasingly hierarchical, heterogeneous, and complex, many research efforts have been made to apply wireless simulations and advanced machine learning techniques for network configuration. Unfortunately, our study shows that the network configuration model generated by the state-of-the-art method decays quickly over time. To address this issue, we develop a *ME*ta-learning based *R*untime *A*daptation (MERA) method that efficiently adapts network configuration models for industrial WSANs at runtime. Under MERA, the parameters of the network configuration model are explicitly trained such that a small number of optimization steps with only a few new measurements will produce good generalization performance after the network condition changes. Experimental results show that MERA achieves higher prediction accuracy with less physical measurements, less computation time, and longer adaptation intervals compared to a state-of-the-art baseline.**

***Index Terms*—IEEE 802.15.4, Industrial Wireless Sensor-Actuator Networks, Runtime Adaptation, Meta-Learning**

I. Introduction

Industrial wireless sensor-actuator networks (WSANs) typically connect sensors, actuators, and controllers in industrial facilities, such as manufacturing plants, steel mills, and oil refineries. IEEE 802.15.4-based wireless networks operate at low-power and can be manufactured inexpensively, which makes them ideal for industrial applications where energy consumption and costs are important. Today a large number of networks that implement IEEE 802.15.4-based industrial WSAN standards, such as WirelessHART [1], ISA100 [2], and 6TiSCH [3], have been deployed in industrial facilities. For instance, Emerson Process Management, one of the leading WirelessHART network suppliers, has deployed more than 54,835 WirelessHART networks globally and gathered 19.7 billion operating hours [4]. A decade of real-world deployments has demonstrated the feasibility of using IEEE 802.15.4-based WSANs to support various industrial applications. However, configuring an industrial WSAN to meet the application-specified quality of service (QoS) requirements is a daunting task, which involves theoretical computation, simulation, and field testing, among other work.

In the literature, significant research efforts have been made to model the characteristics of low-power wireless networks and optimize their configurations by adapting a few physical layer or medium access control (MAC) layer parameters. For instance, Zimmerling et al. [5] developed a framework that helps wireless sensor networks (WSNs) achieve high packet delivery rates by selecting good radio on and off timings in X-MAC [6] and LPP [7] protocols. Peng et al. [8] and Wang et al. [9] proposed to reduce energy consumption by selecting optimal sleep intervals in duty-cycled MAC protocols. As wireless deployments become increasingly hierarchical, heterogeneous, and complex, a breadth of recent research has reported that resorting to advanced machine learning techniques for wireless networking presents significant performance improvements compared to traditional methods. For instance, deep learning is employed to handle a set of parameters for the optimal configurations [10], [11] and reinforcement learning (RL) is used to help the network configure itself [12], [13]. The key behind the performance of those methods is the capability of optimizing a set of free parameters to capture uncertainties, variations, and dynamics in real-world environments. However, it is usually difficult and costly to collect sufficient training data for those data-driven methods in harsh industrial environments. In such scenarios, the benefits of employing the methods that rely on a large amount of physical data are outweighed by the costs. Recently there have been increasing interests in using wireless simulations to identify good configurations for industrial WSANs, because simulations can be set up in less time, introduce less overhead, and allow for different configurations to be tested under the same conditions. However, a recent study showed that the network configurations selected by simulations cannot always help the physical network meet the QoS requirements due to the simulation-to-reality gap [14]. Shi et al. developed a deep learning based domain adaptation method (denoted as DA in this paper) to close the gap. Unfortunately, our study shows that *the network configuration model generated by DA works well at the beginning but decays quickly over time and periodically running DA to update the model introduces too much overhead.*

To address this issue, we develop a *ME*ta-learning based *R*untime *A*daptation (MERA) method that efficiently adapts network configuration models for industrial WSANs at runtime. Under MERA, the parameters of the network configu-

ration model are explicitly trained such that a small number of optimization steps with a small amount of new measurements will produce good generalization performance after the network condition changes. To our knowledge, this paper represents the first study that explores the use of meta-learning for runtime adaptations in industrial WSANs. Specifically, we make the following contributions:

- We present an empirical study to identify the limitations of the state-of-the-art method;
- We formulate the runtime adaptation for industrial WSANs as a machine learning problem and develop a meta-learning based solution, namely MERA;
- We develop a hybrid learning policy (HLP) that helps MERA consistently provide good prediction performance since the physical network starts to operate;
- We implement MERA and evaluate its performance on a testbed that consists of 50 devices. Experimental results show that MERA provides higher prediction accuracy with less physical measurements, less computation time, and longer adaptation intervals compared to our baseline.

The remainder of this paper is organized as follows. Section II presents the background of WirelessHART networks and DA. Section III introduces our empirical study. Section IV presents our design of MERA. Section V describes our experimental evaluation. Section VI reviews the related work. Section VII concludes the paper.

II. Background

In this paper, we use the configuration of WirelessHART networks [1] as an example to present our empirical study and MERA. A WirelessHART network consists of a gateway, multiple access points, and a set of field devices (sensors and actuators). The network manager, a software module running on the gateway, is responsible for the network management including collecting network statistics, generating routes, and scheduling transmissions. To meet the energy efficiency, real-time, and reliability requirements posed by industrial applications, WirelessHART employs the IEEE 802.15.4 physical layer, adopts the time slotted channel hopping (TSCH) technique in the MAC layer, and uses the graph routing in the network layer. Under TSCH, time is split into slices of fixed length that are grouped into a slotframe. All devices are time synchronized and share the notion of a slotframe that repeats over time. Each time slot is long enough to transmit a packet and an acknowledgement between a pair of communicating devices. The network uses up to 16 physical channels in the 2.4GHz ISM band and performs channel hopping in each time slot to combat narrow band interference. WirelessHART networks have three tunable network parameters (the packet reception ratio threshold for link selection R, the number of available physical channels C, and the number of maximum transmission attempts per packet A), which make significant impacts on network performance quantified by three key performance metrics: the end-to-end latency L, the battery lifetime B, and the end-to-end reliability E [14]. The primary goal of network configuration is to select good network parameters (R, C, and A) based on the given QoS requirements (L, B, and E).

It is costly to collect sufficient physical data for data-driven methods in many industrial environments. DA is designed to leverage a large amount of simulation data and a small number of physical measurements to generate good network configuration models for industrial WSANs. Specifically, DA employs deep learning based domain adaptation and leverages a teacher-student neural network to close the simulation-to-reality gap in network configuration. The teacher model is first trained with the simulation data and generates soft labels [15] for the knowledge transfer to the student model. Then the student model is trained with the physical measurements and its parameters are optimized by minimizing the classification loss, domain-consistent loss, and distillation loss simultaneously. To make use of the knowledge learned by the teacher model, the distillation loss is computed with the help of the soft labels.

III. Empirical Study

We have performed an empirical study to investigate the effectiveness and efficiency of DA in identifying good network configurations for WirelessHART networks.

A. Experimental Setup and Data Collection

We adopt the publicly accessible WirelessHART implementation [16] and run experiments on our testbed that consists of 50 TelosB devices placed throughout 22 office and lab areas on the second floor of an office building [17]. We configure the network to have two access points and 48 field devices and set up six data flows with different sources, destinations, data periods, and priorities. There exist 88 distinct network configurations after removing the redundant combinations that lead to the same routes and transmission schedule when considering $R \in \{0.60, 0.61, ..., 0.90\}$, $C \in \{1, 2, ..., 8\}$, and $A \in \{1, 2, 3\}$.

We measure the network performance (L, B, and E) under each of 88 network configurations and save the measurements as a data sample. We define 88 data samples (one data sample under each network configuration) as one shot of data. We collect 15 shots of data (1,320 data samples in total) in each run of experiments. Each run of experiments lasts about four days. We repeat the experiments on our testbed once in every month from August 2021 to March 2022. The measurement collected from our testbed is named as the physical data in this paper. We also implement the same WirelessHART network in the ns-3 simulator [18] and simulate network performance under each of 88 network configurations. We collect 75 shots of simulation data in total.

The physical data collected from each experimental run is split into two disjoint datasets: five shots of data as the training set and 10 shots of data as the testing set. In each experiment, we use 15 shots of physical data and all 75 shots of simulation data. Specifically, we run DA to generate the network configuration model using a training set and 75 shots of simulation data, and then evaluate the model with a testing

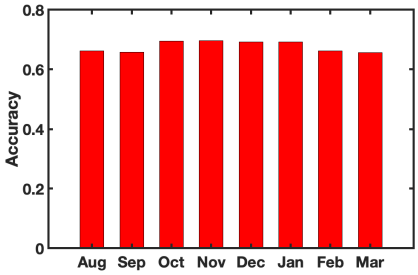


Fig. 1. Prediction accuracy in different months.

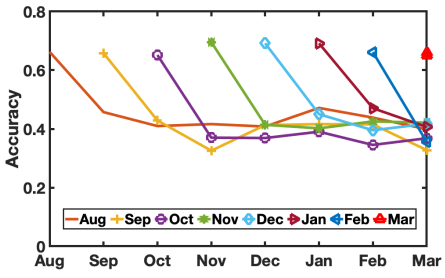


Fig. 2. Accuracy degrades over time.

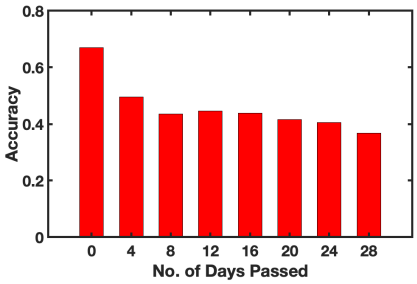


Fig. 3. Accuracy changes over 28 days

set. If the network configuration predicted by the network configuration model based on the application-specified QoS requirements (L, B, and E) is equal to its corresponding label, we define the prediction as a correct prediction. The prediction accuracy is computed by dividing the number of correct predictions by the number of the total testing samples.

B. Performance of DA

We first run DA to generate a network configuration model using the training data collected in each month and measure the prediction accuracy when we use that model to predict network configurations on the testing data collected in that month. Figure 1 plots the prediction accuracy of those eight network configuration models. As Figure 1 shows, the prediction accuracy ranges from 65.65% to 69.59%, which is close to the performance reported by Shi et al. [14].

Observation 1: DA can successfully close the simulation-to-reality gap in network configuration and the model generated by DA can achieve high prediction accuracy.

We further examine whether the performance of DA changes over time. Figure 2 plots the prediction accuracy of the network configuration model generated by DA when it is used for predictions in the following months. As Figure 2 shows, the prediction accuracy provided by the model trained in August decreases from 66.24% to 45.72% when it is tested with the data collected in September and further drops to 40.97% when it is used in October. The model only provides 39.86% accuracy when it is used in March of the following year. Similarly, the network configuration model generated by DA in September provides 65.81% prediction accuracy in the same month, 42.89% accuracy in October, and 32.58% accuracy in November.

To investigate how fast the model decays, we reduce the intervals between our experimental runs and measure the changes on prediction accuracy every four days. As Figure 3 shows, the prediction accuracy begins to decrease after four days and drops significantly from 66.97% to 49.55%. The accuracy further decreases to 43.51% after eight days and drops to 36.81% after 28 days.

Observation 2: The network configuration model produced by DA does not generalize well on new data and decays quickly over time.

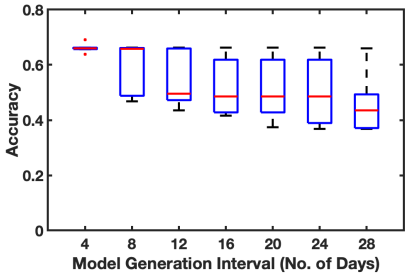


Fig. 4. Prediction accuracy when the network configuration model is updated with different time intervals.

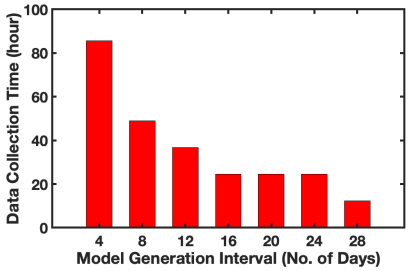


Fig. 5. Overhead over four weeks when the network configuration model is updated with different time intervals.

C. Effectiveness of Runtime Model Updates

Finally, we investigate the feasibility of maintaining high prediction accuracy by periodically running DA to update network configuration model. Figure 4 plots the boxplot of the prediction accuracy when we run DA to generate a new model with different time intervals. As Figure 4 shows, the median accuracy is 66.05% when the model is updated every four days. As a comparison, the median accuracy is 49.47% or 43.53% when the model is updated every 12 or 28 days. The results show that periodically running DA to update the network configuration model can maintain high prediction accuracy.

Please note that the physical network only provides performance measurements under one or more selected configurations. To train a new model, DA must configure the network to operate under other configurations, resulting in undesirable network performance. For example, the end-to-end reliability of the network is 0.3 and the end-to-end latency is $1.44s$, when it uses the configuration ($R = 0.87$, $C = 1$, $A = 2$). Figure 5 plots the time consumed for data collection over four weeks when DA updates the model with different time intervals. As Figure 5 shows, the time consumed by data

collection increases sharply when the model is updated more frequently. To provide 66.05% model prediction accuracy, DA generates seven models during four weeks and spends 85.56 hours to collect sufficient training data from the physical network. The performance degradation during such a long time is unacceptable for most industrial applications.

Observation 3: The amount of the training data required by DA to generate new network configuration models is too large.

Our observations motivate us to develop a new method, which can adapt the network configuration model with less measurements from the physical network.

IV. Design of MERA

In this section, we first formulate the runtime adaptation for industrial WSANs as a machine learning problem and then present our design of MERA.

A. Problem Formulation

The primary goal of runtime adaptation is to help the network meet the application-specified QoS requirements by adapting its configuration at runtime. We assume that u shots of simulation data are gathered from a simulated network before the physical network starts to operate. The simulation data $\mathcal{D}_S$ is evenly divided into m datasets: $\mathcal{D}_{S_1}, \mathcal{D}_{S_2}, ..., \mathcal{D}_{S_m}$. After the physical network starts to operate, the network manager periodically measures the network performance under all configurations and creates the dataset $\mathcal{D}_{P_j}$, where j denotes the j-th time period since the network starts. $\mathcal{D}_{P_j}$ includes v shots of physical data. Our goal is to predict the network configuration, which can help the physical network meet the QoS requirements in the j-th time period, based on the measured physical datasets $\mathcal{D}_{P_1}, \mathcal{D}_{P_2}, ..., \mathcal{D}_{P_j}$ and the simulated datasets $\mathcal{D}_{S_1}, \mathcal{D}_{S_2}, ..., \mathcal{D}_{S_m}$. Therefore, the runtime adaptation for an industrial WSAN can be formulated as a machine learning problem with the goal of learning a nonlinear mapping model $f_\theta(\cdot) : \mathbf{x} \rightarrow \mathbf{y}$ from $\mathcal{D}_{P_1}, \mathcal{D}_{P_2}, ..., \mathcal{D}_{P_j}$ and $\mathcal{D}_{S_1}, \mathcal{D}_{S_1}, ..., \mathcal{D}_{S_m}$, where θ denotes the parameters of f, $\mathbf{x}$ denotes an input vector of the QoS requirements, and $\mathbf{y}$ denotes a vector of network configuration parameters, which can help the network meet the QoS requirements $\mathbf{x}$. The network configuration parameters $\mathbf{y}$ can be discretized without losing generality. Therefore, f_θ can be further restricted as a classifier, which predicts the label of the network configuration $\mathbf{y}$ with the QoS requirements $\mathbf{x}$. As Figure 5 shows, the creation of the physical data $\mathcal{D}_P$ is very costly. Therefore, our goal is to learn a classifier that is robust and can be adapted with the smallest possible $\mathcal{D}_{P_j}$.

B. Overview of MERA

To achieve our goal, we turn our attention to meta-learning, also known as learning to learn, which aims to learn a prior over model parameters that is only a few gradient descent steps away from optimum, enabling fast adaptation to new data using few-shot measurements. The key idea of meta-learning is to train a good model over a variety of learning tasks, each of which is to solve a learning problem (e.g., classification and regression) on a specific dataset, containing both input vectors and true labels. We apply the meta-learning concept into the runtime adaptation for industrial WSANs. Figure 6 shows our design of MERA that consists of four processes: **Offline-Training**, **Online-Training**, **Fine-Tuning**, and **Hybrid Learning Policy (HLP)**. The tasks associated with $\mathcal{D}_S$ for Offline-Training, the tasks associated with $\mathcal{D}_P$ for Online-Training, and the tasks associated with $\mathcal{D}_P$ for Fine-Tuning, are named as simulation tasks, training tasks, and testing tasks, respectively. Before the physical network starts to operate, Offline-Training trains the classifier over m simulation tasks $\mathcal{T}_1^s, \mathcal{T}_2^s, ..., \mathcal{T}_m^s$. After the physical network starts to operate, Online-Training begins to receive the physical datasets from the network manager, initializes the classifier with the parameters θ^0 provided by Offline-Training, and then optimizes the classifier over n training tasks $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n$ to learn a set of good parameters θ^1, which enables the fast adaptation in Fine-Tuning. Fine-Tuning periodically tunes the parameters θ^1 provided by Online-Training based on the latest dataset $\mathcal{D}_{P_j}$ and then predicts the network configuration. The same neural network architecture is used for the classifier in those three learning processes. The finely optimized parameters for predictions are learned by such processes as:

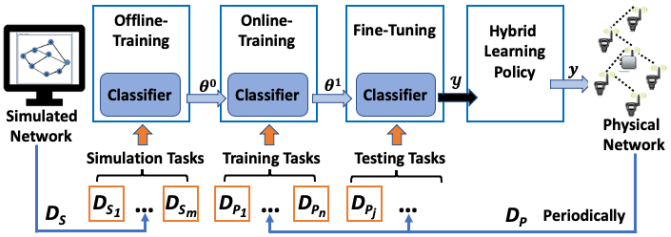


Fig. 6. Overview of MERA.

$$\theta^* = Learn(\mathcal{T}_j; MetaLearn(\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n); PreLearn(\mathcal{T}_1^s, \mathcal{T}_2^s, ..., \mathcal{T}_m^s)) \quad (1)$$

where $PreLearn$, $MetaLearn$, and $Learn$ indicate the parameter optimizations performed by Offline-Training, Online-Training, and Fine-Tuning, respectively.

Meta-learning has proven to be a powerful paradigm for transferring the knowledge from previous tasks to facilitate the learning of a new task, but it may not perform well with insufficient learning tasks [19]. The predicted configurations may lead to poor network performance before the network manager gathers sufficient physical datasets for Online-Training to train a good network configuration model through meta-learning. HLP is designed to address this issue by integrating DA in MERA. We will present the four processes of MERA in detail next.

C. Offline-Training

Offline-Training is designed to speed up Online-Training by training the classifier with the simulation data before the physical network starts to operate. The simulation data shares the same input and label space with the physical data and provides the preliminary knowledge on the features of the

physical data. Specifically, the classifier is optimized over a variety of simulation tasks $\mathcal{T}_1^s, \mathcal{T}_2^s, ..., \mathcal{T}_m^s$ that are associated with the datasets $\mathcal{D}_{S_1}, \mathcal{D}_{S_2}, ..., \mathcal{D}_{S_m}$ and the optimized parameters are computed through:

$$PreLearn(\mathcal{T}_1^s, \mathcal{T}_2^s, ..., \mathcal{T}_m^s) = \arg\min_{\theta} \sum_{i=1}^{m} \mathcal{L}_{\mathcal{T}_i^s}(f_\theta) \quad (2)$$

where m is the number of the simulation tasks and $\mathcal{L}_{\mathcal{T}_i^s}$ is the loss function for the simulation task $\mathcal{T}_i^s$.

Algorithm 1: Offline-Training

Input : s, $\mathcal{D}_{S_1}^{sp}, \mathcal{D}_{S_2}^{sp}, ..., \mathcal{D}_{S_m}^{sp}$, $\mathcal{D}_{S_1}^{qe}, \mathcal{D}_{S_2}^{qe}, ..., \mathcal{D}_{S_m}^{qe}$
Output: θ^0

1 Initialize the classifier randomly;
2 **for** $i = 1; i \leq m; i++$ **do**
3 **for** $p = 1; p \leq s; p++$ **do**
4 Compute $\mathcal{L}_{\mathcal{T}_i^s}(f_{\theta_{p-1}})$ by using $\mathcal{D}_{S_i}^{sp}$ and Eq. 3;
5 Compute updated parameters with gradient descent: $\theta_p = \theta_{p-1} - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i^s}(f_{\theta_{p-1}})$;
6 **end**
7 **end**
8 Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{i=1}^{m} \mathcal{L}_{\mathcal{T}_i^s}(f_{\theta_s})$ by using $\mathcal{D}_{S_i}^{qe}$ and Eq. 3 ($\mathcal{D}_{S_i}^{sp}$ replaced with $\mathcal{D}_{S_i}^{qe}$);
9 Output θ as θ^0;

As meta-learning aims at learning to learn, the classifier is designed to be capable of tackling the unseen tasks through meta-training. To achieve this goal, the simulation data in $\mathcal{D}_{S_i}$ is split into two disjoint parts: support set $\mathcal{D}_{S_i}^{sp}$ and query set $\mathcal{D}_{S_i}^{qe}$. The support set includes l shots of simulation data, while the query set contains k shots of simulation data. The size of $\mathcal{D}_{S_i}^{sp}$ is usually smaller than the size of $\mathcal{D}_{S_i}^{qe}$ ($l < k$), because the classifier is first evaluated on the support set to achieve a set of updated parameters and then the updated classifier is tested with the query set and optimized. Specifically, the loss on the support set of each task $\mathcal{T}_i^s$ takes the following form:

$$\mathcal{L}_{\mathcal{T}_i^s}(f_\theta) = - \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}_{S_i}^{sp}} \mathbf{y} \log f_\theta(\mathbf{x}). \quad (3)$$

Finn et al. [20] showed that there are some internal representations that are more transferable than others and can be discovered by making good use of the training data. Based on this essential idea, the updated parameters θ' is computed by using one or more gradient descent updates on task $\mathcal{T}_i^s$ to quickly adapt to the data samples. For example, when using s gradient descent updates, θ' is denoted as θ_s and computed by using the following functions:

$$\begin{aligned} \theta_0 &= \theta_{init} \\ \theta_1 &= \theta_0 - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i^s}^{(0)}(\theta_0) \\ &\cdots \\ \theta_s &= \theta_{s-1} - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i^s}^{(0)}(\theta_{s-1}) \end{aligned} \quad (4)$$

where α is used to control the updating rate and the superscript in $\mathcal{L}_{\mathcal{T}_i^s}^{(0)}$ indicates the dataset $\mathcal{D}_{S_i}^{sp}$. After computing θ' on each simulation task via s updating steps using $\mathcal{D}_{S_i}^{sp}$, the loss on the query set is computed by using a function, which adopts the form of Eq. 3 but uses $\mathcal{D}_{S_i}^{qe}$ and the updated parameters θ'. Then, the optimization across different simulation tasks is performed via gradient descent using $\mathcal{D}_{S_i}^{qe}$:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{i=1}^{m} \mathcal{L}_{\mathcal{T}_i^s}^{(1)}(f_{\theta_s}) \quad (5)$$

where β is the meta-learning rate and the superscript in $\mathcal{L}_{\mathcal{T}_i^s}^{(1)}$ indicates the dataset $\mathcal{D}_{S_i}^{qe}$. Compared to the standard gradient updating in Eq. 4, the gradient term used in Eq. 5 resorts to a gradient through a gradient that can be named as meta-gradient.

Algorithm 1 shows the detailed procedure of Offline-Training. It first initializes the classifier randomly (line 1) and then performs optimization to update θ (line 2–8). Finally, the parameters θ^0 are provided to Online-Training (line 9).

D. Online-Training

After the network starts to operate, Online-Training initializes the classifier with the parameters θ^0, optimizes the classifier over a variety of training tasks $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n$ that are associated with the physical datasets $\mathcal{D}_{P_1}, \mathcal{D}_{P_2}, ..., \mathcal{D}_{P_n}$, and produces a set of good parameters θ^1 for Fine-Tuning. Specifically, the optimized parameters are learned by executing:

$$MetaLearn(\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n) = \arg\min_{\theta} \sum_{i=1}^{n} \mathcal{L}_{\mathcal{T}_i}(f_\theta) \quad (6)$$

where n is the number of the training tasks and $\mathcal{L}_{\mathcal{T}_i}$ is the loss computed for the training task $\mathcal{T}_i$. Being consistent with Offline-Training, Online-Training reuses Eq. 3-5 for optimization but replaces the simulation tasks $\mathcal{T}_1^s, \mathcal{T}_2^s, ..., \mathcal{T}_m^s$ that are associated with $\mathcal{D}_{S_1}, \mathcal{D}_{S_2}, ..., \mathcal{D}_{S_m}$ with the training tasks $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n$ associated with $\mathcal{D}_{P_1}, \mathcal{D}_{P_2}, ..., \mathcal{D}_{P_n}$. The physical measurements in $\mathcal{D}_{P_i}$ are also split into two disjoint parts: support set $\mathcal{D}_{P_i}^{sp}$ (l shots) and query set $\mathcal{D}_{P_i}^{qe}$ (k shots), to perform meta-training.

Algorithm 2: Online-Training

Input : θ^0, t, $\mathcal{D}_{P_1}^{sp}, \mathcal{D}_{P_2}^{sp}, ..., \mathcal{D}_{P_n}^{sp}$, $\mathcal{D}_{P_1}^{qe}, \mathcal{D}_{P_2}^{qe}, ..., \mathcal{D}_{P_n}^{qe}$
Output: θ^1

1 Initialize the classifier with the parameters $\theta \leftarrow \theta^0$;
2 **for** $i = 1; i \leq n; i++$ **do**
3 **for** $p = 1; p \leq t; p++$ **do**
4 Compute $\mathcal{L}_{\mathcal{T}_i}(f_{\theta_{p-1}})$ by using $\mathcal{D}_{P_i}^{sp}$ and Eq. 3 ($\mathcal{D}_{S_i}^{sp}$ replaced with $\mathcal{D}_{P_i}^{sp}$);
5 Compute updated parameters with gradient descent: $\theta_p = \theta_{p-1} - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta_{p-1}})$;
6 **end**
7 **end**
8 Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{i=1}^{n} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_t})$ by using $\mathcal{D}_{P_i}^{qe}$ and Eq. 3 ($\mathcal{D}_{S_i}^{sp}$ replaced with $\mathcal{D}_{P_i}^{qe}$);
9 Output θ as θ^1;

Algorithm 2 shows the detailed procedure of Online-Training. Instead of using random values, it first initializes the

parameters of the classifier with θ^0 generated by Algorithm 1 (line 1), which speeds up the learning process. It then performs optimization to update θ. Within the nested loop (line 2–7), it iteratively optimizes the parameters learned from the support set through t gradient descent updates. Then, it further optimizes the parameters through meta-gradient by using the query set based on θ_t and updates the classifier with the optimized parameters (line 8). This step (line 2–8) can be executed more than once to make good use of the physical measurements. After multiple iterations of optimization, Online-Training provides the updated parameters θ^1 to Fine-Tuning for its the network configuration predictions.

E. Fine-Tuning

Fine-Tuning is designed to quickly adapt the network configuration model with a few newly collected physical data samples based on a set of good parameters θ^1 and perform well on the genuine testing data. Fine-Tuning only processes one task $\mathcal{T}_j$ at each time. The dataset $\mathcal{D}_{P_j}$ is divided into two parts: support set and query set. The support set $\mathcal{D}^{sp}_{P_j}$ contains the training data used for fine optimization and the query set $\mathcal{D}^{qe}_{P_j}$ contains the genuine QoS requirements $\mathbf{x}$ used to evaluate the performance of the fast adapted classifier. Therefore, the corresponding labels $\mathbf{y}$ are unknown to the classifier.

Algorithm 3: Fine-Tuning

Input : θ^1, r, $\mathcal{D}^{sp}_{P_j}$, $\mathcal{D}^{qe}_{P_j}$
Output: $f_\theta(\mathbf{x})$

1 Initialize the classifier with the parameters $\theta \leftarrow \theta^1$;
2 **for** $p = 1; p \leq r; p++$ **do**
3 Compute $\mathcal{L}_{\mathcal{T}_j}(f_{\theta_{p-1}})$ by using $\mathcal{D}^{sp}_{P_j}$ and Eq. 3 ($\mathcal{D}^{sp}_{S_i}$ replaced with $\mathcal{D}^{sp}_{P_j}$);
4 Compute updated parameters with gradient descent: $\theta_p = \theta_{p-1} - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_j}(f_{\theta_{p-1}})$;
5 **end**
6 Predict the network configuration for the input $\mathbf{x}$ from $\mathcal{D}^{qe}_{P_j}$ using updated parameters θ_r;

Algorithm 3 presents how the classifier is finely tuned with the support set to adapt to each testing task. The classifier is initialized with the parameters θ^1 (line 1). Within the loop, the classifier finishes fast adaptation to a new task with a few new physical data samples through r gradient descent updates (line 2-5). As the query set is used for the actual evaluations, there is no gradient update performed on $\mathcal{D}^{qe}_{P_j}$. Finally, the classifier predicts the corresponding network configuration for each input $\mathbf{x}$ (line 6) and outputs such predictions.

F. HLP

Although meta-learning is known for its high performance on adapting to a new task with few-shot examples, it does not perform well with insufficient training tasks [19]. Therefore, Fine-Tuning may not provide high prediction accuracy when Online-Training fails to identify good parameters because the network manager has not gathered sufficient training datasets since the network starts. HLP is designed to address such an issue by monitoring the network configuration prediction generated by Fine-Tuning and replacing it with the one provided by DA when the latter provides a better prediction. Our implementation of DA adopts the method trained with $\mathcal{D}_{P_j}$ and all simulation datasets.

To minimize the computation overhead introduced by DA (see Section V-B), HLP stops running it after the classifier optimized by Fine-Tuning is capable of helping the network achieve desirable performance at runtime. Specifically, HLP decides whether to run DA based on the increase of the prediction accuracy provided by the classifier generated by Fine-Tuning. The increase of the prediction accuracy becomes very small when the classifier produced by Fine-Tuning is good enough. We define the prediction accuracy increase Δd_j as:

$$\Delta d_j = p_j - p_{j-1} \tag{7}$$

where p_j and p_{j-1} denote the prediction accuracy achieved by Fine-Tuning on $\mathcal{D}^{qe}_{P_j}$ and $\mathcal{D}^{qe}_{P_{j-1}}$, respectively. The prediction accuracy increase averaged among a sliding window (w testing tasks) is computed as:

$$V_j = \frac{1}{w} \sum_{j-w}^{j} \Delta d_i. \tag{8}$$

When the averaged increase V_j exceeds the threshold R_j, HLP runs DA and outputs its prediction. To accommodate network heterogeneity and dynamics, we define R_j as:

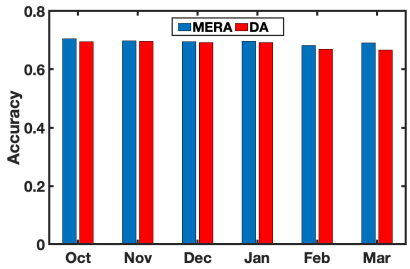
$$R_j = R_{j-1} + \eta(|V_j| - R_{j-1}) \tag{9}$$

where η is a coefficient to determine the change rate of R_j. R_j decreases when the averaged increase of prediction accuracy V_j experiences a gradually decreasing trend. When V_j is smaller than R_j, HLP stops running DA. At the same time, MERA notifies the network manager to suspend the periodical performance measurements (k shots) for Online-Training until V_j exceeds the threshold. Only l shots of physical measurements need to be gathered in each time period for Fine-Tuning to adapt the classifier. This significantly reduces the data collection overhead because $l \ll l + k$.

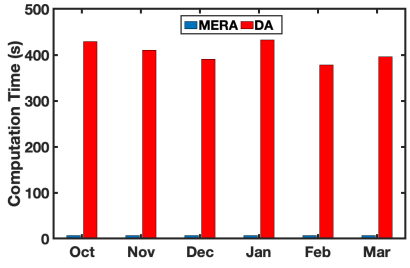
V. Evaluation

We have performed a series of experiments to evaluate MERA. We first perform a six-month experiment that examines the effectiveness and efficiency of MERA in predicting good network configurations at runtime and compares its performance against the one provided by our baseline DA [14] (see Section V-A). We then study the effects of HLP (see Section V-B), the support set size (see Section V-C), and different learning processes (see Section V-D) on MERA's performance. Finally, we evaluate the robustness of the network configuration model generated by MERA (see Section V-E).

We implement MERA and DA under the PyTorch framework [21]. Our implementation employs a deep neural network (DNN) that consists of three fully connected layers for the classifier of MERA. The DNN uses the vector (L, B, E) as the QoS requirements $\mathbf{x}$ and employs 120 neurons and 84 neurons

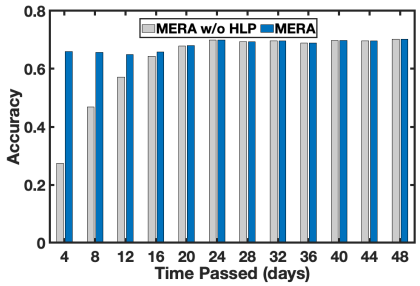


(a) Prediction accuracy.

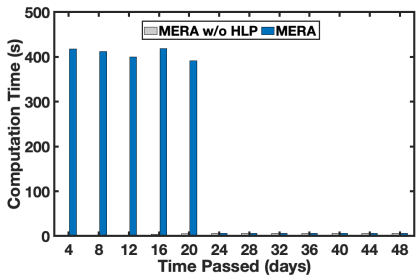


(b) Computation overhead.

Fig. 7. Performance of MERA and DA over six months.



(a) Prediction accuracy.



(b) Computation overhead.

Fig. 8. Effect of HLP.

in the two hidden layers. The number of neurons in its output layer is set to 88, which is equal to the number of distinct network configurations. In addition, the rectified linear unit (ReLU) function and the softmax function are employed to activate the two hidden layers and the output layer, respectively. We use the datasets collected from our testbed (introduced in Section III-A) to create various support and query sets. Each support set includes three shots of data and each query set contains 10 shots of data. Our implementation employs Adam [22] as the optimizer for gradient descent optimization. We empirically set both α and β to 0.01 and the learning rate used in Adam to 0.1 in our experiments. We run MERA and DA on a computer equipped with a 2.6GHz 64-bit hexa-core CPU and an AMD Radeon Pro 5300M GPU.

A. Performance over Six Months

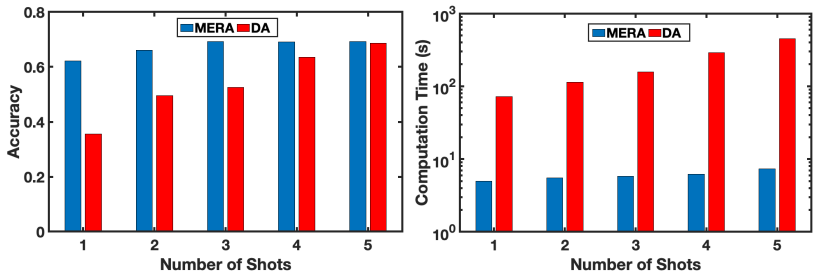
We first examine the effectiveness and efficiency of MERA in predicting good network configurations at runtime over six months and compare its performance against the one provided by DA. The model generated by MERA adapts to a few new measurements (three shots of data) before evaluating with the testing data (10 shots of data) in each month, while DA trains its model with five shots of physical data and the simulation data using its default setting. Figure 7 plots the prediction accuracy and computation overhead when the network runs MERA and DA over six months. As Figure 7(a) shows, the prediction accuracy provided by MERA is around 70%, which is consistently higher than the one provided by DA. For example, the model trained by MERA offers 70.47% prediction accuracy in October, while the model trained by DA provides 69.42% accuracy. Similarly, MERA achieves 69.65% prediction accuracy in January, while the accuracy provided by DA is 69.15%.

Figure 7(b) plots the computation overhead introduced by MERA and DA. As Figure 7(b) shows, the computation time of MERA is about two orders of magnitude less than the execution time of DA. For instance, it takes only $6.31s$ and $6.26s$ to run MERA in October and January, respectively. As a comparison, it takes $429.17s$ and $432.61s$ to run DA in the same months. This is because the training process of DA requires the cross-entropy loss that is optimized many times on a large amount of simulation data together with the physical data, while MERA finely tunes the parameters of the model using only a few iterations with a small number of measurements. The results demonstrate that the model trained by MERA adapts to new observations more efficiently, introducing much less computation overhead. More importantly, it takes 7.33 hours to collect three shots of physical data from the testbed for MERA to achieve such performance, while it consumes 12.22 hours to collect sufficient data for DA to train a model. The results clearly show that MERA provides higher prediction accuracy with significantly less overhead.

B. Effect of HLP

To validate the effectiveness of HLP, we compare the performance of MERA when it enables or disables HLP. As Figure 8(a) shows, the accuracy achieved by MERA without HLP is much lower than the one provided by MERA with HLP during the first few days since the network starts to operate. For example, the accuracy provided by MERA without HLP is 27.50% during the first four days and MERA achieves 65.96% accuracy with the help of HLP. This is because the amount of physical data gathered by the network manager at that time is insufficient for MERA to train a good model. The difference becomes smaller when more physical data is available.

We also measure the training time of MERA when it enables or disables HLP. As Figure 8(b) shows, MERA spends more time during the first 20 days when HLP runs DA. For example, it takes $419.09s$ to run MERA with HLP enabled after 16 days. As a comparison, the time consumption is only $4.08s$ when MERA disables HLP. After HLP observes



(a) Prediction accuracy. (b) Computation overhead.

Fig. 9. Effect of support set size.

Fig. 10. Effects of different processes in MERA.

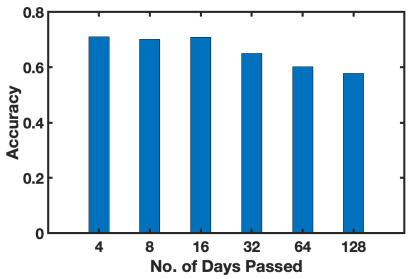


Fig. 11. Prediction accuracy changes over time with Fine-Tuning enabled.

that the increase of the prediction accuracy (V_j in Eq. 8) is 2.18%, less than the threshold 2.39% (R_j in Eq. 9), it stops running DA. The computation time then drops significantly. The results emphasize the importance of stopping running DA when possible.

C. Effect of Support Set Size

The data collection overhead increases with the amount of measurements in the support set used for training. To understand the effect of the support set size on MERA's performance, we vary the amount of measurements in the support set from one shot to five shots and measure the prediction accuracy. Figure 9(a) plots the prediction accuracy provided by MERA and DA when they use different numbers of shots of physical data for training. As Figure 9(a) shows, the prediction accuracy provided by MERA increases from 62.12% to 69.19% when the amount of network measurements increases from one shot to three shots. As a comparison, the model trained by DA achieves 35.56% accuracy when it uses one shot of data and provides 52.49% accuracy when using three shots of data. This is because meta-learning enhances the generalization of the model and helps it achieve fast adaptation with fewer measurements collected from the physical network.

We also measure the time consumed to run MERA and DA when they use different amounts of physical measurements. As Figure 9(b) shows, the computation time of MERA increases when the amount of measurements increases and the time consumed by MERA is always much less than the one used by DA. For instance, MERA spends 4.99s, 5.84s, and 7.33s when it uses one shot, three shots, and five shots for training, respectively. As a comparison, it takes 72.02s, 157.08s, and 447.71s to run DA when it uses one shot, three shots, and five shots for training, respectively. This is because the model trained by MERA that learns through meta-learning can quickly adapt to new measurements with a few iterations, rather than running hundreds of iterations to fit exactly to new observations. The results show that MERA provides high prediction accuracy in a more efficient way, introducing significantly less communication and computation overhead than DA.

D. Effects of Different Learning Processes

To investigate the contributions of MERA's different learning processes to its performance, we run MERA to train a model and measure the prediction accuracy when it disables one or two of its learning processes. As Figure 10 shows, the prediction accuracy decreases to 62.92% without Offline-Training. This clearly shows that valuable network configuration knowledge can be learned from the simulation data even when the simulation-to-reality gap exists. However, only relying on the model generated by Offline-Training for predictions provides 10.16% accuracy, which highlights the importance of closing the simulation-to-reality gap. Online-Training plays the most important role in providing high prediction accuracy. Without Online-Training, the accuracy drops significantly from 71.47% to 27.18%. As a comparison, the model trained by MERA achieves 42.60% accuracy when Online-Training is the only process enabled. When Fine-Tuning is disabled, the accuracy drops to 37.43%. When Fine-Tuning is the only process enabled, MERA provides 3.31% prediction accuracy. The results show that Fine-Tuning also plays an important role in improving the accuracy and relies heavily on other learning processes.

E. Model Robustness

To examine the robustness of the network configuration model trained by MERA, we measure the prediction accuracy after four, eight, 16, 32, 64, and 128 days. As Figure 11 shows, the accuracy values achieved by the network configuration model are 71.03%, 70.21%, and 70.87% after four, eight, and 16 days, respectively. The prediction accuracy degrades slowly over time. The reason behind this is that the model is always finely optimized by Fine-Tuning with a few new measurements.

We repeat the experiments when MERA disables Fine-Tuning. Figure 12 plots the performance achieved by MERA and DA. As Figure 12 shows, the performance degrades under both MERA and DA when the interval increases. However, the performance of the model trained by MERA degrades much

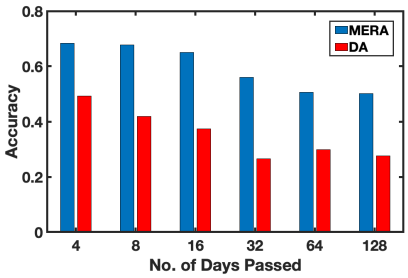


Fig. 12. Prediction accuracy changes over time.

slower. For example, the accuracy values provided by MERA are 68.39%, 65.05%, and 50.08% after four, 16, and 128 days, respectively. As a comparison, the model generated by DA achieves 49.18%, 37.38%, and 27.66% after four, 16, and 128 days. The results demonstrate the robustness of the model generated by MERA. The results also show that this network configuration model does not require frequent adaptations to maintain high accuracy at runtime, which is an important feature of MERA.

VI. Related Work

Early efforts have been made to model the characteristics of low-power wireless networks and optimize their configurations by selecting a few physical layer or MAC layer parameters. For example, Zimmerling et al. [5] developed a framework that automatically optimizes the parameter selections in response to runtime dynamics. Dong et al. proposed to adjust the packet length dynamically to improve energy efficiency [23], [24]. Recently there has been growing interest in leveraging machine learning to configure wireless networks as they become increasingly hierarchical, heterogeneous, and complex. For instance, deep learning based methods are employed to handle a large number of tunable parameters and seek optimal configurations [10], [11] and RL algorithms are adopted to enable network self-configurations [12], [13]. The key behind the success of such methods is the capability of optimizing free parameters to capture extensive uncertainties, variations, and dynamics in real-world environments. However, data collection from industrial facilities that are not easily accessible is very costly. Therefore, it is usually difficult to obtain sufficient physical data to train a good network configuration model by using those data-driven methods. In such scenarios, the benefits of employing those methods that require much physical data are outweighed by the costs. In recent years, online RL solutions [25], [26] have been developed to configure networks at runtime, which introduces less data collection overhead compared to offline RL methods. However, such solutions still follow the trial-and-error principle resulting in undesirable network performance during the learning process.

Recently, there have been increasing interests in using wireless simulations to select the good configurations for industrial WSANs, because simulations can be set up in less time and introduce less overhead. Moreover, various configurations can be tested under the same conditions. However, a recent study showed that a straightforward deployment of the model learned from simulations may result in poor performance in the physical network because of the simulation-to-reality gap [14]. Shi et al. developed a deep learning based domain adaptation method DA to close the gap. Unfortunately, our study shows that the learning model generated by DA works well at the beginning but decays quickly over time and periodically running DA to update the model introduces significant overhead.

Meta-learning [27], [28], aims to solve new learning problems using only a few training samples by leveraging the knowledge learned from a set of related problems. Therefore, it is appealing to few-shot classification problem [29], [30], which evaluates the capability of the system to adapt to new classification tasks with a few examples. Recently, meta-learning algorithms have been widely applied in many areas including computer vision [31], [32], natural language processing [33], [34], and unmanned aerial vehicle (UAV) [35], [36]. In addition, meta-learning algorithms have been explored on other machine learning topics such as reinforcement learning [20], [37]–[39] and federated learning [40], [41]. There are mainly three common approaches to meta-learning: (1) Metric-based methods aim to learn a metric or distance function over objects [29], [42]–[44]; (2) Model-based methods update the parameters with a few steps, which can be achieved by the internal architecture or controlled by another meta-learner model [45], [46]; (3) Optimization-based methods learn an optimized initialization across a set of tasks, allowing fast adaptation to new tasks through one or more updates of gradient descent [20], [30], [47]. Meanwhile, there are a few hybrid studies over these three categories [48], [49]. Among optimization-based methods, MAML [20] has enjoyed the attention for its simplicity and generation performance and been applied in many areas such as clinical risk prediction [50], computer vision [51], frequency division duplexing communication [52]. As MAML is model-agnostic, it is compatible with any model trained with a gradient descent procedure and applicable to a variety of machine learning problems including classification and regression. In this paper, we leverage MAML to develop MERA for industrial WSAN configuration. To our knowledge, this is the first study that explores the use of meta-learning for runtime adaptations in industrial WSANs.

VII. Conclusions

In this paper, we formulate the runtime adaptation for industrial WSANs as a machine learning problem and present MERA to solve the problem. Under MERA, the parameters of the network configuration model are explicitly trained such that a small number of optimization steps with a few new measurements will produce good generalization performance after the network condition changes. We implement MERA and evaluate it on a testbed that consists of 50 devices. Experimental results show that MERA achieves higher accuracy with less physical measurements, less computation time, and longer adaptation intervals compared to a state-of-the-art baseline.

Acknowledgment

This work was supported in part by the National Science Foundation under grant CNS-2150010.

REFERENCES

[1] WirelessHART, "WirelessHART." [Online]. Available: https://www.fieldcommgroup.org/technologies/wirelesshart

[2] ISA100, "ISA100." [Online]. Available: http://www.isa100wci.org/

[3] IETF, "6TiSCH: IPv6 over the TSCH mode of IEEE 802.15.4e." [Online]. Available: https://datatracker.ietf.org/wg/6tisch/documents/

[4] Emerson. [Online]. Available: https://www.emerson.com/en-us/expertise/automation/industrial-internet-things/pervasive-sensing-solutions/wireless-technology

[5] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "PTunes: Runtime Parameter Adaptation for Low-Power MAC Protocols," in *IPSN*, 2012.

[6] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks," in *SenSys*, 2006.

[7] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis, "Koala: Ultra-low Power Data Retrieval in Wireless Sensor Networks," in *IPSN*, 2008.

[8] Y. Peng, Z. Li, D. Qiao, and W. Zhang, "I^2C: A Holistic Approach to Prolong the Sensor Network Lifetime," in *INFOCOM*, 2013.

[9] J. Wang, Z. Cao, X. Mao, and Y. Liu, "Sleep in the Dins: Insomnia Therapy for Duty-cycled Sensor Networks," in *INFOCOM*, 2014.

[10] D. Kumar, T. Amgoth, and C. S. R. Annavarapu, "Machine Learning Algorithms for Wireless Sensor Networks: A Survey," *Information Fusion*, vol. 49, pp. 1–25, 2019.

[11] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[12] S. S. Oyewobi, G. P. Hancke, A. M. Abu-Mahfouz, and A. J. Onumanyi, "An Effective Spectrum Handoff Based on Reinforcement Learning for Target Channel Selection in the Industrial Internet of Things," *Sensors*, vol. 19, no. 6, pp. 1–21, 2019.

[13] F. Li, K.-Y. Lam, Z. Sheng, X. Zhang, K. Zhao, and L. Wang, "Q-Learning-Based Dynamic Spectrum Access in Cognitive Industrial Internet of Things," *Mobile Networks and Applications*, vol. 23, p. 10, 2018.

[14] J. Shi, M. Sha, and X. Peng, "Adapting Wireless Mesh Network Configuration from Simulation to Reality via Deep Learning based Domain Adaptation," in *NSDI*, 2021.

[15] T. Asami, R. Masumura, Y. Yamaguchi, H. Masataki, and Y. Aono, "Domain Adaptation of DNN Acoustic Models using Knowledge Distillation," in *ICASSP*, 2017.

[16] Wireless Cyber-Physical Simulator (WCPS). [Online]. Available: http://wsn.cse.wustl.edu/index.php/WCPS:_Wireless_Cyber-Physical_Simulator

[17] "Testbed at the State University of New York at Binghamton." [Online]. Available: http://users.cs.fiu.edu/%7emsha/testbed.htm

[18] NSNAM, "ns-3 Network Simulator." [Online]. Available: https://www.nsnam.org/

[19] J. Chen, X.-M. Wu, Y. Li, Q. LI, L.-M. Zhan, and F.-l. Chung, "A Closer Look at the Training Strategy for Modern Meta-Learning," in *NeurIPS*, 2020.

[20] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *ICML*, 2017.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *NeurIPS*, 2019.

[22] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015.

[23] W. Dong, C. Chen, X. Liu, Y. He, Y. Liu, J. Bu, and X. Xu, "Dynamic Packet Length Control in Wireless Sensor Networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 3, pp. 1172–1181, 2014.

[24] W. Dong, J. Yu, and P. Zhang, "Exploiting Error Estimating Codes for Packet Length Adaptation in Low-Power Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 14, no. 8, pp. 1601–1614, 2015.

[25] H. Zhang, A. Zhou, J. Lu, R. Ma, Y. Hu, C. Li, X. Zhang, H. Ma, and X. Chen, "OnRL: Improving Mobile Video Telephony via Online Reinforcement Learning," in *MobiCom*, 2020.

[26] H. Mao, M. Schwarzkopf, H. He, and M. Alizadeh, "Towards Safe Online Reinforcement Learning in Computer Systems," in *NeurIPS*, 2019.

[27] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to Learn by Gradient Descent by Gradient Descent," in *NeurIPS*, 2016.

[28] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[29] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra, "Matching Networks for One Shot Learning," in *NeurIPS*, 2016.

[30] S. Ravi and H. Larochelle, "Optimization as a Model for Few-Shot Learning," in *ICLR*, 2017.

[31] D. Li, Y. Yang, Y.-Z. Song, and T. Hospedales, "Learning to Generalize: Meta-Learning for Domain Generalization," in *AAAI*, 2018.

[32] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-Shot Visual Imitation Learning via Meta-Learning," in *CoRL*, 2017.

[33] P.-S. Huang, C. Wang, R. Singh, W.-t. Yih, and X. He, "Natural Language to Structured Query Generation via Meta-Learning," in *NAACL-HLT*, 2018.

[34] F. Mi, M. Huang, J. Zhang, and B. Faltings, "Meta-Learning for Low-resource Natural Language Generation in Task-oriented Dialogue Systems," in *IJCAI*, 2019.

[35] Y. Hu, M. Chen, W. Saad, H. V. Poor, and S. Cui, "Distributed Multi-Agent Meta Learning for Trajectory Design in Wireless Drone Networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 10, pp. 3177–3192, 2021.

[36] B. Li, Z. Gan, D. Chen, and D. Sergey Aleksandrovich, "UAV Maneuvering Target Tracking in Uncertain Environments Based on Deep Reinforcement Learning and Meta-Learning," *Remote Sensing*, vol. 12, no. 22, p. 3789, 2020.

[37] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning," in *CoRL*, 2020.

[38] Z. Xu, H. P. van Hasselt, and D. Silver, "Meta-Gradient Reinforcement Learning," in *NeurIPS*, 2018.

[39] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-Reinforcement Learning of Structured Exploration Strategies," in *NeurIPS*, 2018.

[40] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach," in *NeurIPS*, 2020.

[41] Z. Charles and J. Konečný, "Convergence and accuracy trade-offs in federated learning and meta-learning," in *AISTATS*, 2021.

[42] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature Verification using a "Siamese" Time Delay Neural Network," in *NeurIPS*, 1993.

[43] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to Compare: Relation Network for Few-Shot Learning," in *CVPR*, 2018.

[44] J. Snell, K. Swersky, and R. Zemel, "Prototypical Networks for Few-shot Learning," in *NeurIPS*, 2017.

[45] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-Learning with Memory-Augmented Neural Networks," in *ICML*, 2016.

[46] T. Munkhdalai and H. Yu, "Meta Networks," in *ICML*, 2017.

[47] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson, "Fast Context Adaptation via Meta-Learning," in *ICML*, 2019.

[48] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-Learning with Latent Embedding Optimization," in *ICLR*, 2019.

[49] D. Wang, Y. Cheng, M. Yu, X. Guo, and T. Zhang, "A Hybrid Approach with Optimization-based and Metric-based Meta-Learner for Few-Shot Learning," *Neurocomputing*, vol. 349, pp. 202–211, 2019.

[50] X. S. Zhang, F. Tang, H. H. Dodge, J. Zhou, and F. Wang, "MetaPred: Meta-Learning for Clinical Risk Prediction with Limited Patient Electronic Health Records," in *KDD*, 2019.

[51] M. Choi, J. Choi, S. Baik, T. H. Kim, and K. M. Lee, "Scene-Adaptive Video Frame Interpolation via Meta-Learning," in *CVPR*, 2020.

[52] J. Zeng, J. Sun, G. Gui, B. Adebisi, T. Ohtsuki, H. Gacanin, and H. Sari, "Downlink CSI Feedback Algorithm With Deep Transfer Learning for FDD Massive MIMO Systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 4, pp. 1253–1265, 2021.