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

XIA CHENG and MO SHA*, Florida International University, USA

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. We also develop a data sampling method to reduce the measurements required by MERA at runtime without sacrificing its performance. 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.

CCS Concepts: • **Networks → Wireless local area networks**; **Network management**; **Network simulations**; **Network performance modeling**; **Network measurement**; • **Computing methodologies → Machine learning approaches**.

Additional Key Words and Phrases: IEEE 802.15.4, Industrial Wireless Sensor-Actuator Networks, Runtime Adaptation, Meta-Learning

ACM Reference Format:
Xia Cheng and Mo Sha. 2023. MERA: Meta-Learning Based Runtime Adaptation for Industrial Wireless Sensor-Actuator Networks. *ACM Trans. Sensor Netw.* X, X, Article X (January 2023), 23 pages. https://doi.org/10.1145/1122445.1122456

1 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 [47], ISA100 [20], and 6TiSCH [19], have been deployed in

*Corresponding author

Part of this article was published in Proceedings of the IWQoS.

Authors' address: Xia Cheng; Mo Sha, Florida International University, 11200 SW 8th St, Miami, FL, 33199, USA, {xchen075, msha}@fiu.edu.

1550-4859/2023/1-ARTX $15.00
https://doi.org/10.1145/1122445.1122456

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 [11]. 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. [55] 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 [5] and LPP [31] protocols. Peng et al. [35] and Wang et al. [45] 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 [23, 51] and reinforcement learning (RL) is used to help the network configure itself [27, 33]. 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 [40]. 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 configuration 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 develop a data sampling method to effectively and efficiently reduce the physical measurements required by MERA at runtime without sacrificing its prediction performance;

- 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 2 presents the background of WirelessHART networks and DA. Section 3 introduces our empirical study. Section 4 presents our design of MERA. Section 5 introduces our data sampling method. Section 6 describes our experimental evaluation. Section 7 reviews the related work. Section 8 concludes the paper.

2 BACKGROUND

In this paper, we use the configuration of WirelessHART networks [47] 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 [40]. 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 [2] 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.

3 EMPIRICAL STUDY

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

3.1 Experimental Setup and Data Collection

We adopt the publicly accessible WirelessHART implementation [46] and run experiments on our testbed that consists of 50 TelosB devices placed throughout 22 office and lab areas on the second

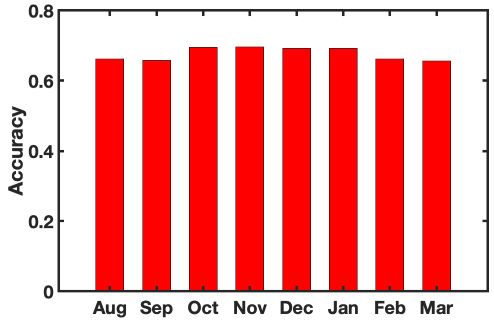


Fig. 1. Prediction accuracy in different months.

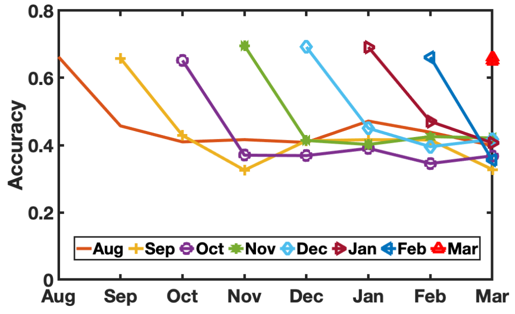


Fig. 2. Accuracy degrades over time.

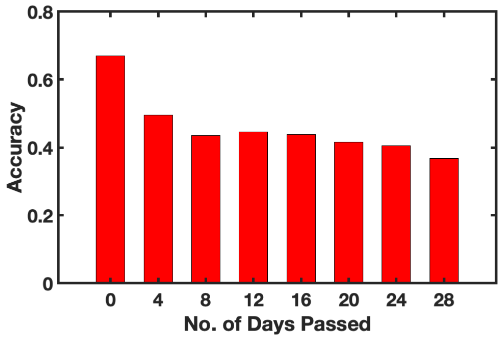


Fig. 3. Accuracy changes over 28 days.

floor of an office building [39]. 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 such data collection 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 [32] 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 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.

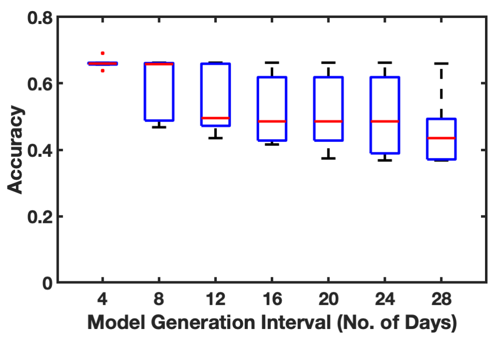


Fig. 4. Prediction accuracy when the network configuration model is updated with different time intervals.

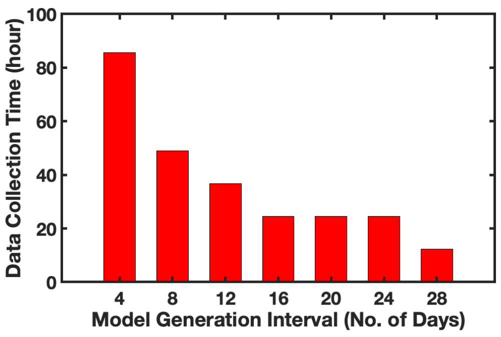


Fig. 5. Overhead over four weeks when the network configuration model is updated with different time intervals.

3.2 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. [40].

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.

3.3 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.

4 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.

4.1 Problem Formulation

The primary goal of runtime adaptation is to help the network meet the application-specified performance 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 performance 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 network performance requirements, and $\mathbf{y}$ denotes a vector of network configuration parameters, which can help the network meet the performance 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 performance requirements as the input $\mathbf{x}$. 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 $\mathcal{D}_{P_j}$.

4.2 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.

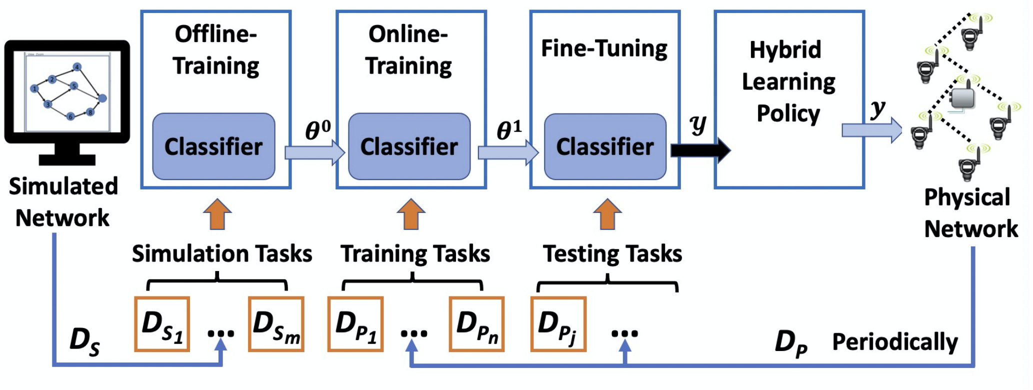


Fig. 6. Overview of MERA.

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:

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

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 [7]. 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.

4.3 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:

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

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 : $\theta^0$
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 $\theta$ as $\theta^0$;
```

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:

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

Finn et al. [13] 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:

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

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 $\mathcal{D}_{S_i}^{qe}$:

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

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).

4.4 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:

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

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  : $\theta^0$, $t$, $\mathcal{D}_{P_1}^{sp}, \mathcal{D}_{P_2}^{sp}, ..., \mathcal{D}_{P_n}^{sp}, \mathcal{D}_{P_1}^{qe}, \mathcal{D}_{P_2}^{qe}, ..., \mathcal{D}_{P_n}^{qe}$
Output : $\theta^1$
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 $\theta$ as $\theta^1$;
```

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.

4.5 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 $\mathcal{D}_{P_j}^{sp}$ contains the training data used for fine optimization and the query set $\mathcal{D}_{P_j}^{qe}$ 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  : $\theta^1$, $r$, $\mathcal{D}_{P_j}^{sp}$, $\mathcal{D}_{P_j}^{qe}$
  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 $\mathcal{D}_{P_j}^{sp}$ and Eq. 3 ($\mathcal{D}_{S_i}^{sp}$ replaced with $\mathcal{D}_{P_j}^{sp}$);
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 $\mathcal{D}_{P_j}^{qe}$ using updated parameters $\theta_r$;
```

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 $\mathcal{D}_{P_j}^{qe}$. Finally, the classifier predicts the corresponding network configuration for each input $\mathbf{x}$ (line 6) and outputs such predictions.

4.6 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 [7]. 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 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 as the traditional method.

To minimize the computation overhead introduced by the traditional method (see Section 6.2), 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 the traditional learning method 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 the traditional learning method 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 the traditional learning method. 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 $l \ll l + k$.

5 PHYSICAL DATA SAMPLING

In this section, we first present our empirical study to quantify the distribution discrepancy among the physical datasets gathered in different time periods. Then, we introduce a data sampling method to identify the physical data samples needed to be collected for Fine-Tuning. The primary goal is to reduce the data collection overhead without significantly degrading the quality of the network configuration model.

5.1 Quantify Data Discrepancy

The network performance measurements gathered by the network manager vary when the network condition changes. To compare the physical data collected in different time periods, it is beneficial to quantify the distribution discrepancy among them. The network performance measurements $\mathbf{x}$ consist of different types of physical values, which vary among different ranges. For example, the end-to-end latency ranges from $0.05s$ to $2.0s$, while the end-to-end reliability of the network varies from 0.02 to 1.0. Therefore, we preprocess the physical data by performing the standardization process based on the following function:

$$x^* = \frac{x - \mu}{\sigma}. \tag{10}$$

where μ is the mean value of the physical data, and σ is the standard deviation of the data. After the standardization process, all performance measurements vary among the range between zero and one.

Next, we adopt the maximum mean discrepancy (MMD) criterion proposed in [3] to quantify the discrepancy among different physical datasets. MMD has been widely used in various machine learning algorithms to compare distributions without the prior knowledge on their density functions. The key idea of MMD is to measure the distance between two distributions based on the mean of features in the reproducing kernel Hilbert space (RKHS) after mapping them to RKHS. Specifically, MMD can be computed by following the equation:

$$MMD(X,Y) = \left\| \frac{1}{N_X}\sum_{i=1}^{N_X} \phi(x_i) - \frac{1}{N_Y}\sum_{j=1}^{N_Y} \phi(y_i) \right\|_H^2 \tag{11}$$

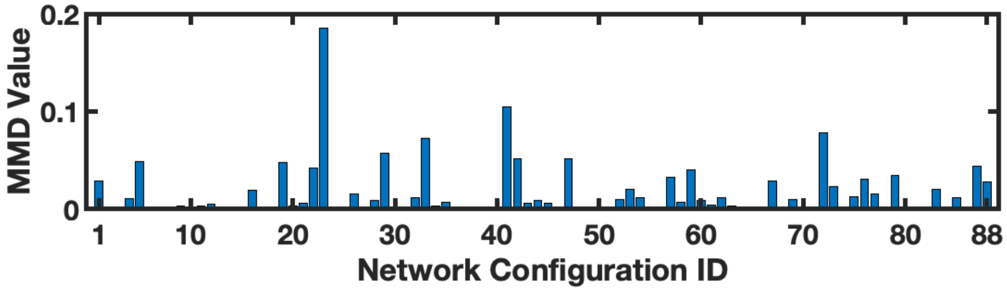


Fig. 7. Discrepancy between $\mathcal{D}_1$ and $\mathcal{D}_2$.

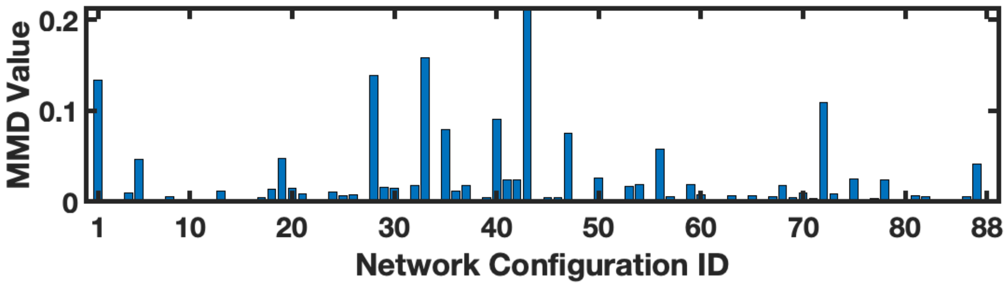


Fig. 8. Discrepancy between $\mathcal{D}_1$ and $\mathcal{D}_3$.

where X and Y represent two different physical datasets, N_X and N_Y indicate the numbers of data samples in X and Y, respectively, ϕ is the function that maps X and Y to RKHS, and H represents RKHS.

Figure 7 plots the discrepancy between two physical datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, which are collected at different times, under each of 88 distinct network configurations (IDs range from 1 to 88). As Figure 7 shows, the MMD value varies sharply under different network configurations. For example, the MMD values are 0.185, 0.104, and 0.078, under Network Configuration 23, 41, and 72, respectively. As a comparison, the MMD values are 3.50×10^{-5}, 6.34×10^{-5}, and 8.80×10^{-5}, under Network Configuration 14, 49, and 64, respectively. The small values close to zero indicate that the two physical datasets share similar distributions under many network configurations.

Figure 8 plots the distribution difference between $\mathcal{D}_1$ and $\mathcal{D}_3$ under each network configuration. The MMD value is up to 0.213 under Network configuration 43, while the value is only 2.26×10^{-6} under Network Configuration 52. More importantly, compared to the results shown in Figure 7, the MMD values are consistently small under many network configurations. For instance, the MMD value varies slightly from 1.10×10^{-2} to 1.01×10^{-2} under Network Configuration 4. Similarly, the value decreases from 8.48×10^{-5} to 7.44×10^{-5} under Network Configuration 31. Such comparisons clearly show that the network performance changes slightly under those network configurations, leading to the similar or even identical distributions among three different datasets.

5.2 Data Sampling Method

Collecting training data from a physical network not only leads to performance degradation discussed in Section 3.3, but also consumes a significant amount of energy and time. As Table 1 shows, it consumes 1,864J and takes 7.33 hours to collect three shots of data from a physical network with 50 devices. The energy and time consumption increases significantly when collecting more data for training. Therefore, it is important to reduce the number of physical data samples collected for training a good network configuration model. As shown in Figure 7 and Figure 8, the network

Table 1. Data Collection Overhead

Number of Shots	1	3	5	10	15
Energy Consumption (J)	621	1,864	3,150	6,228	9,319
Collection Time (hour)	2.44	7.33	12.22	24.44	36.67

performance measurements do not change or change slightly under many configurations. Inspired by this observation, we develop a data sampling method to reduce the physical data collected periodically for Fine-Tuning, without significantly sacrificing the network configuration prediction accuracy. The key idea is to suspend the physical data collection under the configurations where the discrepancy quantified by the MMD criterion between the datasets gathered in different time periods is small.

Algorithm 4 presents how to determine the network configurations where the network manager continues gathering the performance measurements, based on the comparison between the historical data and the newly collected data. The input of Algorithm 4 consists of three parameters: the historical physical dataset $\mathcal{D}_P^H$, the physical dataset $\mathcal{D}_{P_j}$ that includes all physical data samples that have already been collected by the network manager in the j-th time period, and the historical MMD values m^H. Algorithm 4 first extracts the latest dataset $\mathcal{D}_C$ from $\mathcal{D}_P^H$ (line 1) and then regularizes the data in $\mathcal{D}_C$ and $\mathcal{D}_{P_j}$ (line 2). Within the loop, it traverses each network configuration, computes the MMD value m_j^i under the i-th configuration according to Eq. 11, and updates the historical record m^H (line 4). When the network performance varies dramatically but the gathered dataset $\mathcal{D}_{P_j}$ shares a similar distribution with $\mathcal{D}_C$ by chance, m_j^i may not represent the discrepancy effectively. To exclude such corner cases, we define the exponential weighted average value M_j as follows,

$$M_j = \frac{1}{z} \sum_{j-z}^{j} e^{-(j-x+1)} m_x^i. \tag{12}$$

where z is the size of the sliding window (the number of time periods). After computing M_j^i (the value of M_j under the i-th network configuration), Algorithm 4 decides to stop the data collection under the i-th network configuration in the j-th time period, if M_j^i is no larger than the threshold M_R (line 7). Otherwise, it requests the network manager to continue collecting the performance measurements under the i-th network configuration (line 10). We have performed a series of ablation studies to identify the best-suited value for M_R and set M_R to 0.08 in our implementation (see Section 6.6). The time complexity of computing the MMD value is $O(N_X N_Y)$ [21, 54], where N_X and N_Y indicate the numbers of data samples under one network configuration in $\mathcal{D}_C$ and $\mathcal{D}_{P_j}$, respectively. Accordingly, the time complexity of Algorithm 4 is $O(M N_X N_Y)$, where M is the number of distinct network configurations.

6 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 [40] (see Section 6.1). We then study the effects of HLP (see Section 6.2), the support set size (see Section 6.3), and different learning processes (see Section 6.4) on MERA's performance. Finally, we evaluate the robustness of the network configuration model generated by MERA (see Section 6.5) and study the effect of our data sampling method on the performance of MERA (see Section 6.6).

We implement MERA and DA under the PyTorch framework [34]. Our implementation employs a deep neural network (DNN) that consists of three fully connected layers for the classifier of

Algorithm 4: Data Sampling Method

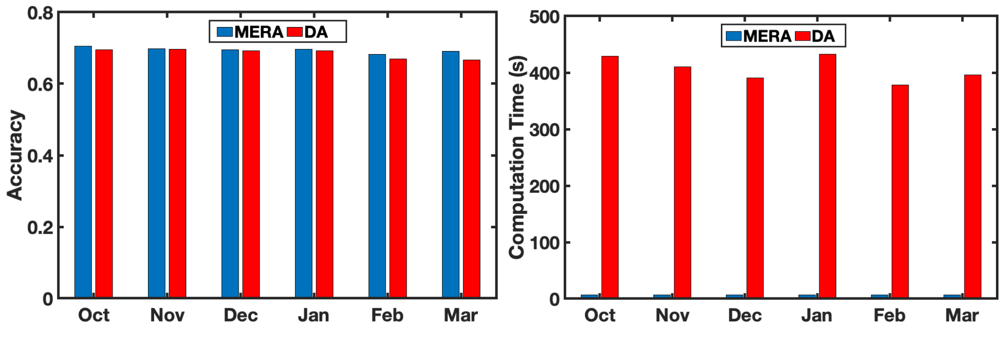
```
   Input  : $\mathcal{D}_P^H$, $\mathcal{D}_{P_j}$, $m^H$
1  Extract the latest dataset $\mathcal{D}_C$ from $\mathcal{D}_P^H$, where $\mathcal{D}_C$ and $\mathcal{D}_{P_j}$ share the same size;
2  Regularize the data in $\mathcal{D}_C$ and $\mathcal{D}_{P_j}$;
3  for $i = 1; i \leq M; i++$ do
4      Compute MMD value $m_j^i$ under the $i$-th network configuration between $\mathcal{D}_C$ and $\mathcal{D}_{P_j}$
         and record;
5      Compute $M_j^i$ based on Eq. 12;
6      if $M_j^i <= M_R$ then
7          Decide to stop data collection under the $i$-th configuration in the $j$-th period;
8      end
9      else
10         Continue data collection for this configuration;
11     end
12 end
```

MERA. The DNN uses the vector (L, B, E) as the QoS requirements $\mathbf{x}$ and employs 120 neurons and 84 neurons 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 3.1) to create various support and query sets. All experiments are performed with the same settings for the data flows. 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.

6.1 Performance over Six Months

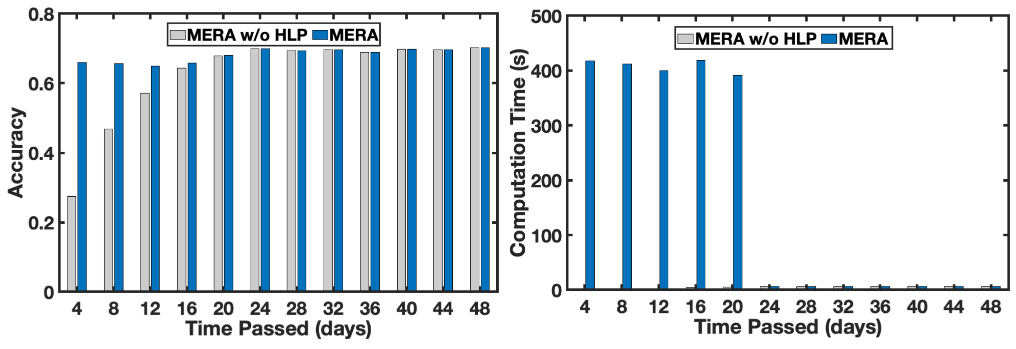
We first examine the effectiveness and efficiency of MERA in predicting good network configurations at runtime over six months and compare its performance against the one provided by DA. The model generated by MERA adapts to a few new measurements (three shots of data) before evaluating with the testing 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. We define the prediction as a correct one if the network configuration predicted by the model generated by MERA based on the input (L, B, and E) is equal to its corresponding label, and calculate the prediction accuracy by dividing the number of correct predictions by the total number of testing samples. Figure 9 plots the prediction accuracy and computation overhead when the network runs MERA and DA over six months. As Figure 9(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 9(b) plots the computation overhead introduced by MERA and DA. As Figure 9(b) shows, the computation time of MERA is about two orders of magnitude less than the execution time of DA.



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

Fig. 9. Performance of MERA and DA over six months.



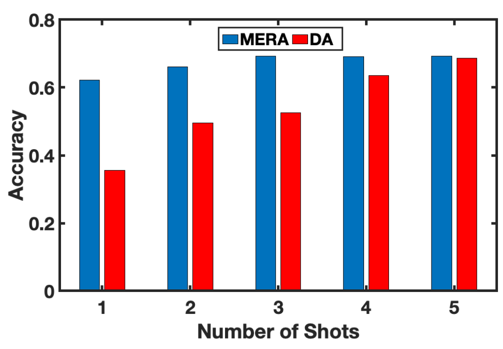
(a) Prediction accuracy. (b) Computation overhead.

Fig. 10. Effect of HLP.

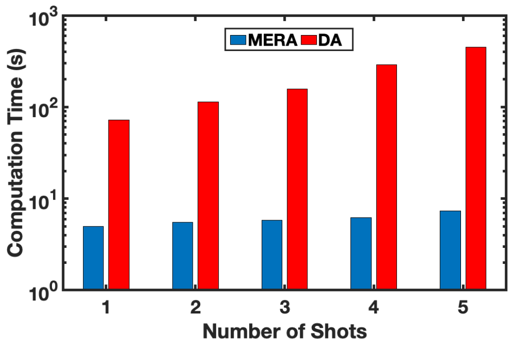
For instance, it takes only 6.31*s* and 6.26*s* to run MERA in October and January, respectively. As a comparison, it takes 429.17*s* and 432.61*s* to run DA in the same months. This is because the training process of DA requires the cross-entropy loss that is optimized many times on a large amount of 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.

6.2 Effect of HLP

To validate the effectiveness of HLP, we compare the performance of MERA when it enables or disables HLP. As Figure 10(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.



(a) Prediction accuracy.



(b) Computation overhead.

Fig. 11. Effect of support set size.

We also measure the training time of MERA when it enables or disables HLP. As Figure 10(b) shows, MERA spends more time during the first 20 days when HLP runs the traditional method. 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 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 the traditional method. The computation time then drops significantly. The results emphasize the importance of stopping running the traditional method when possible.

6.3 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 11(a) plots the prediction accuracy provided by MERA and DA when they use different numbers of shots of physical data for training. As Figure 11(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 11(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.

6.4 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

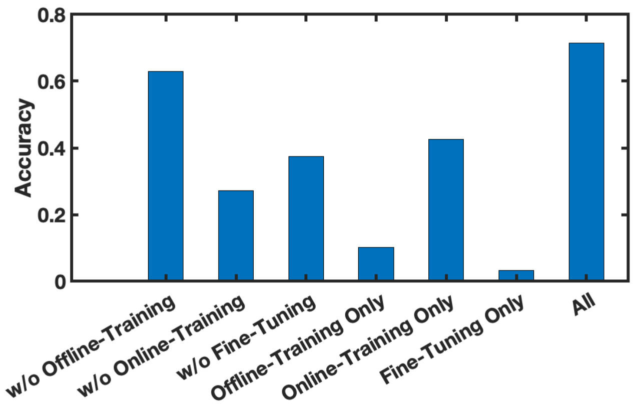


Fig. 12. Effects of different processes in MERA.

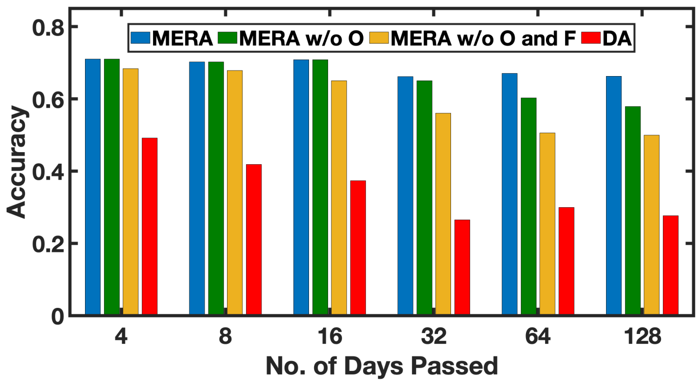


Fig. 13. Prediction accuracy changes over time (O – Online-Training; F – Fine-Tuning).

learning processes. As Figure 12 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.

6.5 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 the blue bars in Figure 13 show, the prediction accuracy degrades slowly over time. For instance, the accuracy values achieved by the network configuration model are 71.03%, 70.87%, and 67.02% after four, 16, and 64 days, respectively. The results demonstrate the robustness of the model generated by MERA.

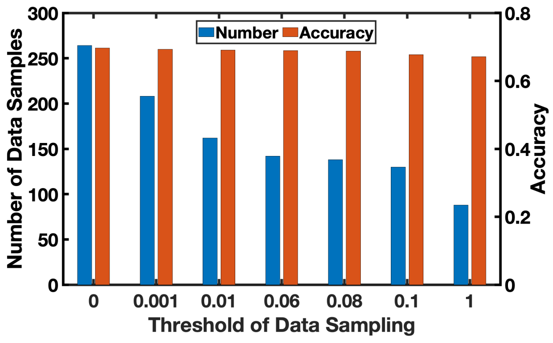


Fig. 14. Number of data samples and accuracy changes with the threshold of data sampling.

We repeat the experiments when MERA disables Online-Training. As the green bars in Figure 13 show, MERA achieves significantly lower prediction accuracy after 64 and 128 days. For example, MERA provides 60.28% prediction accuracy after 64 days without the help of Online-Training. The comparison demonstrates the importance of Online-Training in consistently providing high prediction accuracy at runtime.

We also measure the prediction accuracy provided by MERA without Online-Training and Fine-Tuning. The yellow and red bars in Figure 13 plot the performance achieved by MERA and DA, respectively. The performance degrades under both MERA and DA when the interval increases. However, the performance of the model trained by MERA degrades much 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 show that this network configuration model does not require frequent adaptations to maintain high accuracy at runtime, which is an important feature of MERA.

6.6 Effects of Data Sampling

Finally, we examine the effectiveness and efficiency of our data sampling method on reducing the data collection overhead. We first study the effect of the threshold M_R on the performance of our data sampling method. Figure 14 plots the numbers of data samples determined by our data sampling method in November and the corresponding accuracy offered by MERA, when M_R varies from zero to one and the sliding window size is set to one. As Figure 14 shows, both the number of data samples and the accuracy decrease with the increasing M_R. After M_R is larger than 0.08, the prediction accuracy experiences a rapid drop. For instance, the prediction accuracy achieved by MERA using 138 data samples at 0.08 is 68.80%, while the accuracy provided by MERA using 130 data samples at 0.1 is 67.72%. We observe similar trends when repeating the experiments on other datasets, thus we decide to keep the threshold at 0.08 and the sliding window at one in the rest of our experiments.

We use our data sampling method to select samples from the data traces used in the experiments presented in Section 6.1. Figure 15 plots the numbers of data samples collected for Fine-Tuning when we run MERA with or without our data sampling method. As Figure 15 shows, the numbers of data samples required by Fine-Tuning to optimize the network configuration model are significantly reduced with the help of our data sampling method. For instance, the network manager is required to collect 264 physical data samples (three shots of data) in December and March without the help of Algorithm 4. As a comparison, the number of data samples collected for training drops to 120 in December and 136 in March, when our data sampling method is running. Meanwhile, MERA provides comparable prediction accuracy in these months, as shown in Figure 16. For example,

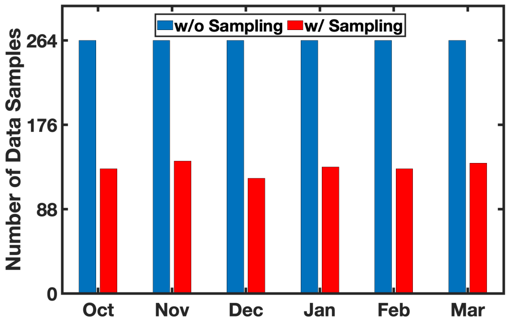


Fig. 15. Data Reduction.

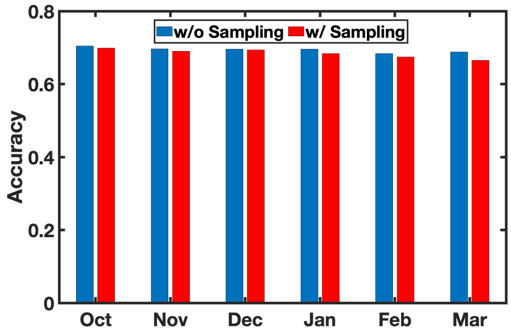


Fig. 16. Prediction accuracy.

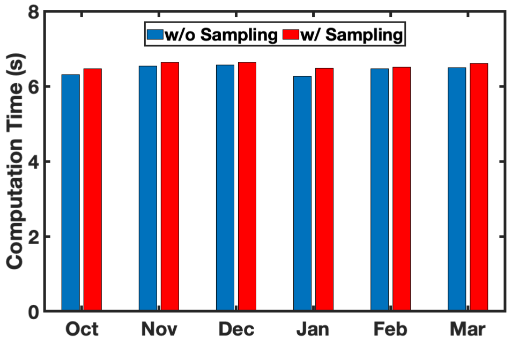


Fig. 17. Computation overhead.

MERA achieves 69.57% prediction accuracy in December using 264 data samples, while it provides 69.38% prediction accuracy using 120 data samples with the help of our data sampling method. Similarly, the prediction accuracy achieved by MERA decreases slightly from 68.37% to 67.47% in February. The results demonstrate that our data sampling method slightly degrades the network prediction performance in exchange for a proportionally much larger reduction in the amount of physical data required to be collected by the network manager.

We also measure the execution time of MERA when we enable our data sampling method and compare it to the running time without our data sampling method. As Figure 17 shows, MERA spends slightly more time in each month when it enables our data sampling method. For instance, MERA spends 6.31s without running Algorithm 4 in October, while it spends 6.46s using Algorithm 4 in the same month. Similarly, the execution time increases slightly from 6.56s to 6.63s in December. The results clearly show that our data sampling method reduces the data collection overhead without introducing significant computation overhead. Therefore, it is beneficial to employ MERA together with our data sampling method to perform runtime adaptation for industrial wireless networks where the communication overhead is a major concern.

7 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. [55] 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 [9, 10]. Fu et al. highlighted the challenges of adapting multiple parameters simultaneously because of their joint effect on performance [15]. 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 the optimal configurations [23, 51] and RL algorithms are adopted to enable network self-configurations [27, 33]. The key behind the success of those learning methods is the capability of optimizing a series of 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 [28, 52] 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 [40]. 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 [1, 24], 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 [36, 43], 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 [14, 26], natural language processing [18, 29], and unmanned aerial vehicle (UAV) [17, 25]. In addition, meta-learning algorithms have been explored on other machine learning topics such as reinforcement learning [13, 16, 48, 49] and federated learning [6, 12]. There are mainly three common approaches to meta-learning: (1) Metric-based methods aim to learn a metric or distance function over objects [4, 41–43]; (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 [30, 38]; (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 [13, 36, 56]. Meanwhile, there are a few hybrid studies over these three categories [37, 44]. Among optimization-based methods, MAML [13] has enjoyed the attention for its simplicity and generation performance and been applied in many areas such as clinical risk prediction [53], computer vision [8], frequency division duplexing communication [50]. 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.

8 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 also develop a data sampling method to reduce the number of physical measurements required for training without sacrificing the prediction performance of MERA. 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] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. 2016. Learning to Learn by Gradient Descent by Gradient Descent. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 3988–3996.

[2] Taichi Asami, Ryo Masumura, Yoshikazu Yamaguchi, Hirokazu Masataki, and Yushi Aono. 2017. Domain Adaptation of DNN Acoustic Models using Knowledge Distillation. In *ICASSP*. IEEE, USA, 5185–5189.

[3] Karsten M. Borgwardt, Arthur Gretton, Malte J. Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J. Smola. 2006. Integrating Structured Biological Data by Kernel Maximum Mean Discrepancy. *Bioinformatics* 22, 14 (2006), e49–e57.

[4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification using a "Siamese" Time Delay Neural Network. In *NeurIPS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 737–744.

[5] M. Buettner, G. V. Yee, E. Anderson, and R. Han. 2006. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. In *SenSys*. Association for Computing Machinery, New York, NY, USA, 307–320.

[6] Zachary Charles and Jakub Konečný. 2021. Convergence and Accuracy Trade-Offs in Federated Learning and Meta-Learning. In *AISTATS*. PMLR, 2575–2583.

[7] Jiaxin Chen, Xiao-Ming Wu, Yanke Li, Qimai LI, Li-Ming Zhan, and Fu-lai Chung. 2020. A Closer Look at the Training Strategy for Modern Meta-Learning. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 396–406.

[8] Myungsub Choi, Janghoon Choi, Sungyong Baik, Tae Hyun Kim, and Kyoung Mu Lee. 2020. Scene-Adaptive Video Frame Interpolation via Meta-Learning. In *CVPR*. IEEE Computer Society, Los Alamitos, CA, USA, 9441–9450.

[9] Wei Dong, Chun Chen, Xue Liu, Yuan He, Yunhao Liu, Jiajun Bu, and Xianghua Xu. 2014. Dynamic Packet Length Control in Wireless Sensor Networks. *IEEE Transactions on Wireless Communications* 13, 3 (2014), 1172–1181.

[10] Wei Dong, Jie Yu, and Pingxin Zhang. 2015. Exploiting Error Estimating Codes for Packet Length Adaptation in Low-Power Wireless Networks. *IEEE Transactions on Mobile Computing* 14, 8 (2015), 1601–1614.

[11] Emerson. 2020. Emerson. https://www.emerson.com/en-us/expertise/automation/industrial-internet-things/pervasive-sensing-solutions/wireless-technology

[12] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2020. Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 3557–3568.

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*. JMLR.org, 1126–1135.

[14] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. 2017. One-Shot Visual Imitation Learning via Meta-Learning. In *CoRL*. PMLR, 357–368.

[15] Songwei Fu, Yan Zhang, Yuming Jiang, Chengchen Hu, Chia-Yen Shih, and Pedro Jose Marron. 2015. Experimental Study for Multi-layer Parameter Configuration of WSN Links. In *ICDCS*. IEEE Computer Society, Los Alamitos, CA, USA, 369–378.

[16] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. 2018. Meta-Reinforcement Learning of Structured Exploration Strategies. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 5307–5316.

[17] Ye Hu, Mingzhe Chen, Walid Saad, H. Vincent Poor, and Shuguang Cui. 2021. Distributed Multi-Agent Meta Learning for Trajectory Design in Wireless Drone Networks. *IEEE Journal on Selected Areas in Communications* 39, 10 (2021), 3177–3192.

[18] Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural Language to Structured Query Generation via Meta-Learning. In *NAACL-HLT*. Association for Computational Linguistics, New Orleans, Louisiana, 732–738.

[19] IETF. 2022. 6TiSCH: IPv6 over the TSCH mode of IEEE 802.15.4e. https://datatracker.ietf.org/wg/6tisch/documents/

[20] ISA100. 2018. ISA100. http://www.isa100wci.org/

[21] Beomjoon Kim and Joelle Pineau. 2013. Maximum Mean Discrepancy Imitation Learning. In *Robotics: Science and systems*.

[22] Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[23] D.Praveen Kumar, Tarachand Amgoth, and Chandra Sekhara Rao Annavarapu. 2019. Machine Learning Algorithms for Wireless Sensor Networks: A Survey. *Information Fusion* 49 (2019), 1–25.

[24] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338.

[25] Bo Li, Zhigang Gan, Daqing Chen, and Dyachenko Sergey Aleksandrovich. 2020. UAV Maneuvering Target Tracking in Uncertain Environments Based on Deep Reinforcement Learning and Meta-Learning. *Remote Sensing* 12, 22 (2020), 3789.

[26] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. 2018. Learning to Generalize: Meta-Learning for Domain Generalization. In *AAAI*. AAAI Press, 3490–3497.

[27] Feng Li, Kwok-Yan Lam, Zhengguo Sheng, Xinggan Zhang, Kanglian Zhao, and Li Wang. 2018. Q-Learning-Based Dynamic Spectrum Access in Cognitive Industrial Internet of Things. *Mobile Networks and Applications* 23 (2018), 10.

[28] Hongzi Mao, Malte Schwarzkopf, Hao He, and Mohammad Alizadeh. 2019. Towards Safe Online Reinforcement Learning in Computer Systems. In *NeurIPS*.

[29] Fei Mi, Minlie Huang, Jiyong Zhang, and Boi Faltings. 2019. Meta-Learning for Low-resource Natural Language Generation in Task-oriented Dialogue Systems. In *IJCAI*. AAAI Press, 3151–3157.

[30] Tsendsuren Munkhdalai and Hong Yu. 2017. Meta Networks. In *ICML*. JMLR.org, 2554–2563.

[31] Razvan Musaloiu-E., Chieh-Jan Mike Liang, and Andreas Terzis. 2008. Koala: Ultra-low Power Data Retrieval in Wireless Sensor Networks. In *IPSN*. IEEE Computer Society, USA, 421–432.

[32] NSNAM. 2022. ns-3 Network Simulator. https://www.nsnam.org/

[33] Stephen S. Oyewobi, Gerhard P. Hancke, Adnan M. Abu-Mahfouz, and Adeiza J. Onumanyi. 2019. An Effective Spectrum Handoff Based on Reinforcement Learning for Target Channel Selection in the Industrial Internet of Things. *Sensors* 19, 6 (2019), 1–21.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 8026–8037.

[35] Yang Peng, Zi Li, Daji Qiao, and Wensheng Zhang. 2013. I^2C: A Holistic Approach to Prolong the Sensor Network Lifetime. In *INFOCOM*. IEEE, USA, 2670–2678.

[36] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a Model for Few-Shot Learning. In *ICLR*.

[37] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. 2019. Meta-Learning with Latent Embedding Optimization. In *ICLR*.

[38] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-Learning with Memory-Augmented Neural Networks. In *ICML*. JMLR.org, 1842–1850.

[39] Mo Sha. 2022. Testbed at the State University of New York at Binghamton. http://users.cs.fiu.edu/%7emsha/testbed.htm

[40] Junyang Shi, Mo Sha, and Xi Peng. 2021. Adapting Wireless Mesh Network Configuration from Simulation to Reality via Deep Learning based Domain Adaptation. In *NSDI*. USENIX Association, 887–901.

[41] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical Networks for Few-shot Learning. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 4080–4090.

[42] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales. 2018. Learning to Compare: Relation Network for Few-Shot Learning. In *CVPR*. IEEE, USA, 1199–1208.

[43] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. 2016. Matching Networks for One Shot Learning. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 3637–3645.

[44] Duo Wang, Yu Cheng, Mo Yu, Xiaoxiao Guo, and Tao Zhang. 2019. A Hybrid Approach with Optimization-based and Metric-based Meta-Learner for Few-Shot Learning. *Neurocomputing* 349 (2019), 202–211.

[45] Jiliang Wang, Zhichao Cao, Xufei Mao, and Yunhao Liu. 2014. Sleep in the Dins: Insomnia Therapy for Duty-cycled Sensor Networks. In *INFOCOM*. IEEE, USA, 1186–1194.

[46] WCPS. 2023. Wireless Cyber-Physical Simulator (WCPS). http://wsn.cse.wustl.edu/index.php/WCPS:_Wireless_Cyber-Physical_Simulator

[47] WirelessHART. 2022. WirelessHART. https://www.fieldcommgroup.org/technologies/wirelesshart

[48] Zhongwen Xu, Hado P van Hasselt, and David Silver. 2018. Meta-Gradient Reinforcement Learning. In *NeurIPS*. Curran Associates Inc., Red Hook, NY, USA, 2402–2413.

[49] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. 2020. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *CoRL*. 1094–1100.

[50] Jun Zeng, Jinlong Sun, Guan Gui, Bamidele Adebisi, Tomoaki Ohtsuki, Haris Gacanin, and Hikmet Sari. 2021. Downlink CSI Feedback Algorithm With Deep Transfer Learning for FDD Massive MIMO Systems. *IEEE Transactions on Cognitive Communications and Networking* 7, 4 (2021), 1253–1265.

[51] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2224–2287.

[52] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. 2020. OnRL: Improving Mobile Video Telephony via Online Reinforcement Learning. In *MobiCom*. Association for Computing Machinery, New York, NY, USA, 1–14.

[53] Xi Sheryl Zhang, Fengyi Tang, Hiroko H. Dodge, Jiayu Zhou, and Fei Wang. 2019. MetaPred: Meta-Learning for Clinical Risk Prediction with Limited Patient Electronic Health Records. In *KDD*. Association for Computing Machinery, New York, NY, USA, 2487–2495.

[54] Ji Zhao and Deyu Meng. 2015. FastMMD: Ensemble of Circular Discrepancy for Efficient Two-Sample Test. *Neural Computation* 27, 6 (2015), 1345–1372.

[55] Marco Zimmerling, Federico Ferrari, Luca Mottola, Thiemo Voigt, and Lothar Thiele. 2012. PTunes: Runtime Parameter Adaptation for Low-Power MAC Protocols. In *IPSN*. Association for Computing Machinery, New York, NY, USA, 173–184.

[56] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. 2019. Fast Context Adaptation via Meta-Learning. In *ICML*. PMLR, 7693–7702.