# RestoreML: Practical Unsupervised Tuning of Deployed Intelligent IoT Systems

Jinyang Li*, Yizhuo Chen*, Ruijie Wang, Tomoyoshi Kimura, Tianshi Wang, You Lyu, Hongjue Zhao, Binqi Sun†, Shangchen Wu, Yigong Hu, Denizhan Kara, Beitong Tian, Klara Nahrstedt, Suhas Diggavi‡, Jae H Kim§, Greg Kimberly§, Guijun Wang§, Maggie Wigness¶, Tarek Abdelzaher
Email: tarek@illinois.edu
*University of Illinois Urbana-Champaign*, USA
†*Technical University of Munich*, Germany
‡*University of California, Los Angeles*, USA
§*Boeing Research & Technology*, USA
¶*DEVCOM Army Research Laboratory*, USA

***Abstract*—As today's edge AI systems age, the need for incremental updates of *intelligent* sensor nodes (that fail or reach the end of their useful lifetime) becomes a growing concern. Replacement nodes may happen to use new versions of sensors, may have different physical properties (e.g., inertia or stiffness), or may run updated signal processing firmware, thereby changing the characteristics of sensor waveforms (such as acoustic, seismic, or acceleration measurements) made available to the downstream AI model. New downloadable AI models might be less than a perfect fit because they might be trained in an environment that differs from the conditions of the old deployment. As a result of such sensor and/or AI differences, new replacement nodes may not perform optimally when deployed. *They will need to be fine-tuned after deployment*. Recognizing this problem, this paper introduces RestoreML, a novel algorithm designed to fine-tune AI models in an *unsupervised* manner (i.e., without the need for labeling or human intervention). The algorithm leverages advances in *test-time adaptation* (TTA) to refine machine-learning (ML) models with *unlabeled* data collected after deployment. Innovations are introduced in the way deployment data are sampled for model fine-tuning, and the way less reliable nodes are automatically identified. RestoreML is implemented as a middleware library and a broker that can be easily integrated into existing applications. Evaluation results on a specially-curated dataset, M3N-VC (that we make publicly available[1]), demonstrate that RestoreML can significantly enhance model performance after deployment, especially in node replacement scenarios, outperforming state-of-the-art baselines.**

***Index Terms*—Test-time Adaptation, Multi-Node IoT Sensing, Vehicle Monitoring, Knowledge Distillation**

## I. Introduction

The paper addresses an often-overlooked problem in today's increasingly AI-driven IoT landscape – namely, the challenge of reducing the logistic cost of AI that arises when it is time to replace failed or malfunctioning *intelligent* nodes.

Replacement of *traditional* sensor or IoT nodes, such as standard fire alarms, requires only functional compatibility of the used sensors. In contrast, when sensors are connected to a downstream *edge AI component* (possibly running elsewhere in the system, such as on an edge server), replacing or upgrading the sensor may cause subtle incompatibilities with the downstream AI. Even if a new AI model is downloaded to match the new sensor, it may have subtle differences in training that make it less compatible with the old deployment environment. Such subtle incompatibilities, arising from the replacement, may degrade intelligent system performance. To restore good performance, model fine-tuning may be needed. Ordinarily, fine-tuning requires labeled data and, thus, human intervention. Instead, this paper explores an *unsupervised* approach.

We cast this challenge as a special case of the *domain shift* problem [1]. *Domain shift* broadly refers to a discrepancy between the data distribution in the deployment environment (*target domain*) and that of the training dataset (*source domain*). *Domain adaptation* commonly refers to adapting the models to the shifted target domains [2]–[4]. When the adaptation algorithm has no access to labeled data from the target domain, the problem is often referred to as *test-time adaptation* (TTA) [5]. Our intelligent IoT node replacement problem constitutes a challenging category of TTA problems, where (i) the *target domain data are not accessible in the training phase* (e.g., because the original training has no access to the characteristics of future replacement sensors), (ii) the available *target domain data are unlabeled* (no manual labeling in the field), and (iii) the *source domain data are not available upon deployment* (we might not have access to the training data of the original node's vendor). Prior TTA methods addressed this problem using techniques such as minimizing prediction entropy [6], adapting normalization layers [7], or performing data augmentation [8]. However, as will be empirically demonstrated in Section V, these methods exhibit limited performance in our IoT application scenario because they do not exploit the multi-view nature of the IoT system and often require well-calibrated uncertainty estimation, which is hard to ensure for IoT models with limited training data.

Instead, the novel contribution of RestoreML lies in exploiting three common characteristics of IoT systems to improve

* These authors contributed equally to this work.
[1]Out dataset M3N-VC is publicly available at: https://github.com/restoreml/m3n-vc and https://doi.org/10.5281/zenodo.15215210

over existing TTA solutions: (1) IoT systems are often multi-node or multi-view, offering multiple observations of the same event. (2) Since wholesale infrastructure replacements are relatively rare, incremental updates are more common, which ultimately increases the diversity of deployed node versions, generations, and technologies, allowing for solutions that leverage diversity to improve performance. (3) The monitored events tend to persist spatially and temporally, allowing AI predictions to be calculated over an appropriate window in space and time.

Accordingly, our solution is novel in exploiting correlations in distributed measurements, exploiting the diversity of nodes used, and leveraging the persistence of stimuli over time and space to design TTA mechanisms that do not require well-calibrated uncertainty estimation. More specifically, we design RestoreML as a *collaborative knowledge distillation* technique [9] that operates by collecting and weighing predictions made by individual node models within a given spatial-temporal window and uses the resulting ensemble prediction as a soft label to fine-tune the individual models that need fine-tuning. The technique is novel in two respects. First, RestoreML uses a new algorithm for deciding how deployment data are sampled for model fine-tuning purposes that has a crucial effect on performance by minimizing bias and reducing the chances of catastrophic forgetting. Second, it is optimized to quickly accommodate and "teach" new nodes, which requires automated means for identifying misbehaving models to avoid ensemble contamination with results from poorly tuned sensors in the absence of well-calibrated uncertainty estimation.

To empirically evaluate our method, allow reproducibility, and facilitate future IoT research for the community, we release our experimental data and algorithms. Our experimental data are shared in the form of a large-scale novel IoT vehicle monitoring dataset (18.26 hours, 31.25 GB), which we call **M3N-VC**, (**M**ulti-**M**odality **M**ulti-**N**ode **V**ehicle **C**lassification). M3N-VC consists of data collected in six different environments. We use 6–8 nodes in each environment to collect seismic and acoustic signals for multiple moving vehicles. The dataset supports a variety of research topics, including domain adaptation, multi-node pretraining, multi-node tracking, and vehicle classification, among others.

Evaluation results demonstrate that, for the applications and datasets considered in this study, our approach generally improves model performance after deployment and achieves superior performance compared to several existing state-of-the-art test-time adaptation baselines. In summary, the contributions of this paper are three-fold:

- We propose RestoreML, a novel collaborative knowledge distillation-based TTA method customized for intelligent IoT node replacement scenarios.
- We collect and release M3N-VC, a new large-scale real-world multi-modality multi-node vehicle monitoring dataset collected in diverse environments for our own testing and to fuel future multi-node IoT sensing and domain adaptation research.
- The evaluation reveals RestoreML's superior adaptation performance.

The rest of this paper is organized as follows. Section II reviews related work. Section III presents some background, followed by the design details of RestoreML in Section IV. The evaluation is presented in Section V. The paper concludes with Section VI.

## II. Related Work

**Test-Time Adaptation (TTA)** is a sub-topic of Domain Adaptation [2], [3], aiming at adapting a model pre-trained on the source domain to the target domain. Unlike Continual Domain Adaptation [10]–[13], TTA solely depends on unlabeled target data without accessing the source data or target data labels. Generative models such as CGAN [14] and VAE [15] were adopted for enhancing the model adaptation capability [16], [17] in test-time domain style. Wang et al. [6] proposed to minimize the entropy of the predictions on target data and only adjust the affine transformation parameters of the batch normalization layer. Zhang et al. [8] extended this approach by minimizing the entropy of augmentation-averaged predictions during test time. Wulfmeier et al. [18] justified and unified entropy minimization with other test-time losses by viewing them as optimizing with respect to a pseudo-label, uniquely defined for each source domain training loss. Recalculating the parameters of the batch normalization layers to account for covariance shift in the target domain was also introduced [7], [19]. This idea was extended by taking batch normalization parameters learned from the source domain as a prior for the target domain batch normalization parameters [20]. Kingma et al. [21] proposed to assign pseudo-labels for target domain data and finetune the model with the pseudo-labels. However, these methods primarily rely on single-node information and overlook the multi-node nature of IoT applications. Moreover, they are mainly designed for computer vision tasks with large-scale datasets and assume models that offer well-calibrated uncertainty estimates, which is difficult to guarantee in IoT applications with limited training data. In contrast, our method makes no assumptions on model calibration and leverages the unique opportunities in multi-node IoT applications, making it particularly well-suited for IoT deployments.

**Collaborative Knowledge Distillation**, or Mutual Learning, aims at training multiple neural networks simultaneously and transferring knowledge between them to boost performance. Knowledge distillation was proposed to ensemble from multiple models to enhance classification accuracy [22], [23]. Built upon this, Zhu et al. [24] proposed to gather multiple models into a single multi-branch neural network to facilitate feature sharing and dynamically adjust the weight of the ensemble using a gate neural network. Chen et al. [25] followed the ensemble framework, with an additional leader model distilled from the ensemble of all peers. Guo et al. [9] further systematically studied various ensemble policies specific to mutual learning scenarios with the help of existing ground truth labels. Li et al. [26] proposed to combine mutual distillation with a temporal distillation from an ensemble of previous model

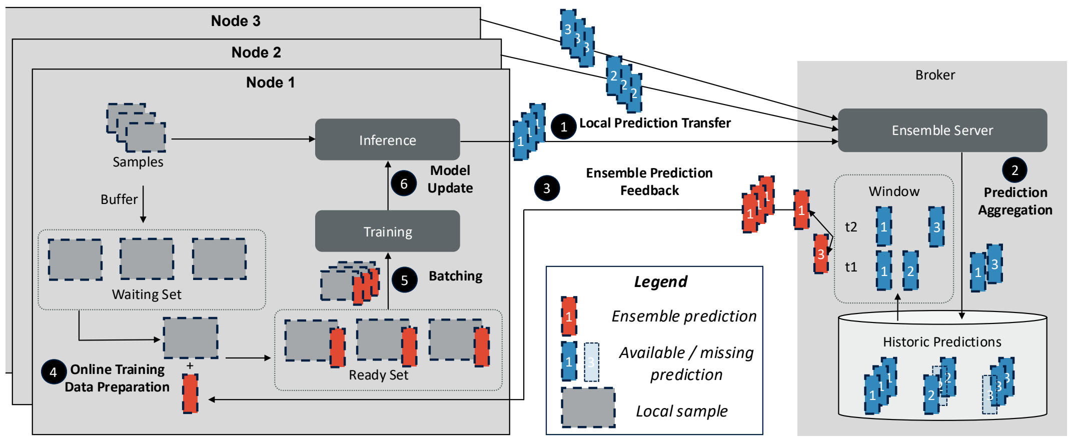

Fig. 1. An overview of collaborative knowledge distillation, where critical steps are marked with numbers and are described in detail in Section III.

snapshots to the current model as well as an entropy minimization loss for semi-supervised learning tasks. Wu et al. [27] proposed to adopt a meta-model-based ensemble policy in mutual learning and combined it with temporal distillation. Li et al. [28] proposed to introduce a proxy teacher distilled from the original teacher that is also simultaneously distilled mutually with the student model to enhance the performance of the student model. However, these collaborative knowledge distillation methods are proposed not for domain adaptation purposes, but for enhancing the performance of traditional supervised learning. In contrast, our method customizes collaborative knowledge distillation for TTA without annotations for IoT node replacement scenarios.

Similar to our method, Zhang et al. [29] proposed an ensemble method based on the Dempster–Shafer theory that is scaled using a confidence score to generate soft labels for online training. However, their work is designed for *in-domain* data labeling and thus requires access to source domain training data, which is not available in our case.

Also, we differ from federated learning [30]–[34] in that we aim to train per-node *local models* in an *unsupervised* setting, whereas federated learning aims to train a single *global model* in a *supervised* setting.

## III. Background

To explain the workings of our algorithm, we first review the general category of collaborative knowledge distillation approaches and then introduce the specifics of our approach. An overview of general collaborative distillation systems is depicted in Figure 1.

Consider a system that consists of $N$ nodes ($N > 1$), where each node is an IoT device equipped with single or multiple sensors continually monitoring the environment. To perform intelligent sensing tasks, data from each node is processed by one or more machine learning models, where the $j$-th model running on data from the $i$-th node is denoted as $\theta_{i,j}$. Let there be at most $M$ models per node.

Each node receives a stream of samples, where $\boldsymbol{x}_{i,t}$ denotes the sample collected at node $i$ and time $t$. In time-series data, a sample refers to data collected within a set time interval. Each $\boldsymbol{x}_{i,t}$ can be single- or multi-modal, depending on the sensors used. Each model $\theta_{i,j}$ makes a inference instantly after the $\boldsymbol{x}_{i,t}$ is received, which can be written as

$$\hat{\boldsymbol{y}}_{i,j,t} = f(\boldsymbol{x}_{i,t}; \theta_{i,j}), \tag{1}$$

where $\hat{\boldsymbol{y}}_{i,j,t}$ denotes the prediction, $f(\boldsymbol{x}_{i,t}\theta_{i,j})$ denotes the execution of model $\theta_{i,j}$ on $\boldsymbol{x}_{i,t}$.

After the inference, each node sends its predictions to a logically centralized node, we henceforth call *the broker*. As shown in ❷, the broker, upon receiving the individual node predictions, provides a spatial-temporal ensemble prediction, denoted as $\bar{\boldsymbol{y}}_{i,t}$. Importantly, the ensemble prediction $\bar{\boldsymbol{y}}_{i,t}$ can be different for each node when our system has different ground truth labels for different nodes. For example, if multiple vehicles are running in a nearest-vehicle classification system, each node may have its own different ground truth (a different nearest vehicle) and, thus, different individual predictions and ensemble predictions.

As described in ❸, the ensemble prediction $\bar{\boldsymbol{y}}_{i,t}$ is then sent back to each node (that needs to be fine-tuned), serving as a soft label to fine-tune its model. The ensemble prediction $\bar{\boldsymbol{y}}_{i,t}$ is calculated as a weighted average of all predictions made from: 1) all models $j \in \{1, \ldots, M\}$; 2) all nodes inside of a neighboring spatial window of node $i$, denoted as $\mathcal{S}_i$; and 3) all times inside of temporal window of time $t$, denoted as $\mathcal{T}_t$, which can be written as:

$$\bar{\boldsymbol{y}}_{i,t} = \frac{1}{Z_{i,t}} \sum_{j' \in \{1,\ldots,M\},\ i' \in \mathcal{S}_i,\ t' \in \mathcal{T}_t} w_{i',j',t'} \hat{\boldsymbol{y}}_{i',j',t'}, \tag{2}$$

where $w_{i',j',t'}$ denotes the ensemble weight of the individual prediction $\hat{\boldsymbol{y}}_{i',j',t'}$. $Z_{i,t}$ is a normalization factor calculated as the sum of all participated wights:

$$Z_{i,t} = \sum_{j' \in \{1,\ldots,M\},\ i' \in \mathcal{S}_i,\ t' \in \mathcal{T}_t} w_{i',j',t'}. \tag{3}$$

Therefore, $\bar{\boldsymbol{y}}_{i,t}$ is guaranteed to fall into the probability simplex.

As illustrated in ❹, each node then locally logs the $\bar{\boldsymbol{y}}_{i,t}$ received from the broker and its associated sample $\boldsymbol{x}_{i,t}$. Each node periodically runs an online model updating process to update its ML models. In the updating process for model $\theta_{i,j}$, according to ❺, node $i$ first fetches a batch of $(\boldsymbol{x}_{i,t}, \bar{\boldsymbol{y}}_{i,t})$ pairs from its local storage, denoted as $\mathcal{R}_{i,j}$. Then, as depicted in ❻, the node calculates a loss function on $\mathcal{R}_{i,j}$, denoted as $\mathcal{L}_{i,j}$, to update $\theta_{i,j}$ with stochastic gradient descent:

$$\theta'_{i,j} \leftarrow \theta_{i,j} - \eta \nabla_\theta \mathcal{L}_{i,j}, \tag{4}$$

where $\eta$ is the learning rate. $\theta'_{i,j}$ is the updated model, which should be more accurate in the target domain. $\theta'_{i,j}$ would then be used to calculate future predictions and ensembles, which could reciprocally improve the accuracy of the ensemble model, forming a positive feedback loop. Note that, ground truth is not assumed to be known in deployment and is not used by the algorithm.

## IV. System Design

Building on the above generic description of collaborative knowledge distillation (adapted in a straightforward manner to the IoT domain), it remains to answer two central questions that distinguish different approaches and constitute the core of our contribution. First, since training performance depends largely on the choice of data used for training, how to select the batches, $\mathcal{R}_{i,j}$, from the data logged in each round of online updates? Poor selection can result in bias, catastrophic forgetting, and other learning inefficiencies. A well-suited answer to the data selection question is a core differentiator of our algorithm. Second, since training performance also depends largely on the choice of nodes entrusted with contributions to (soft) data labeling, how to automatically distinguish adequate models from misbehaving ones, in the absence of supervision and in the absence of model confidence information? Failure to do so will result in poor training performance and, thus, prediction quality degradation. These questions are addressed in the following subsections, respectively.

### *A. Least-trained-balanced Sampling Policy*

This subsection answers the following key question: which data should be included in a given batch, $\mathcal{R}_{i,j}$, for a more effective AI model update?

One seemingly obvious approach might be to use the most recently collected data since that data would constitute the new observations not previously trained with. The problem with this approach is that it does not ensure that the samples used in the batch are independent. The most recent samples are likely collected around the same time and are thus typically correlated. For example, they might feature the same targets or similar environmental conditions. Learning, in contrast, requires that samples included with a batch be independent and identically distributed (iid), drawn from a distribution that sufficiently represents all conditions and targets of interest. Failure to do so can result in problems such as a bias for specific classes or catastrophic forgetting, where updates that optimize the neural network for new conditions damage or erase the knowledge of other previously learned conditions.

The most straightforward training data selection policy that ensures an iid training data distribution in a batch is *random sampling*. The policy randomly chooses a batch of samples from all locally stored ones. Unfortunately, random sampling results in unsatisfactory performance as well. Its inadequacy is attributed to two different reasons. First, the data received in real-life scenarios can be highly unbalanced (for example, most vehicles might be of a particular class). The random sampling policy will repetitively update the individual models on the unbalanced data, which may cause the individual models to get gradually biased toward the popular classes while impairing recognition of other classes. Second, using a random sampling policy, samples received earlier are chosen more often than those received later, potentially making the models overfit on earlier samples and consequently unable to classify the rest of the data robustly.

To solve these problems, we introduce the *least-trained-balanced* sampling strategy to enhance the training stability and the utility of the logged data. Specifically, we propose to assign each $(\boldsymbol{x}_{i,t}, \bar{\boldsymbol{y}}_{i,t})$ pair to the class whose probability in $\bar{\boldsymbol{y}}_{i,t}$ is the largest, and maintain a counter of how many times each $(\boldsymbol{x}_{i,t}, \bar{\boldsymbol{y}}_{i,t})$ pair is trained in previous online updating rounds. When we need to assemble a batch of data, we choose an equal number of samples for each class and make sure that, within each class, the chosen samples are least trained in previous rounds. As we show in the evaluation section, under the *least-trained-balanced* sampling strategy, our individual models are updated with samples chosen from all classes and at all times equally, resulting in model performance improvements.

In practice, when sensor hardware, signal processing chains, or AI models undergo significant updates, our current scheme resets the model's data reservoir. This reset triggers a fresh adaptation cycle, allowing the system to adapt specifically to the characteristics introduced by the updated configuration. Future work could further refine this strategy by dynamically adjusting the temporal distribution of samples post-reset, prioritizing recent or significantly changed operational conditions to accelerate model adaptation.

### *B. Automatic Ensemble Weights Determination*

Another important choice in our system design is to determine the ensemble weights $w_{i',j',t'}$ for all nodes, such that predictions of less confident models are weighted lower. Unfortunately, as mentioned earlier, accurate estimation of model confidence requires good model calibration, which is often hard to achieve. Thus, a key design decision of our algorithm (and one of its key advantages) is *not to require confidence estimation*. Instead, weights are assigned *in the absence of confidence information*. A straightforward policy to do so is to assign equal weights for all participating models, which has been shown to deliver satisfactory empirical performance and desirable theoretical properties in some ML literature [35], [36]. However, this all-equal weighting policy has limitations in our scenario, where some individual models may suffer from drastic performance degradation in the target domain – for example, when some models receive data from an updated/replaced sensor that differs from the ones used in their original training. These unreliable models should be automatically identified, and their contributions should be eliminated from the ensemble calculations for better training accuracy. Ideally, since we do not know if a sensor (or other component) replacement has caused problems with downstream AI, and since we would like to accommodate other reasons for AI quality deterioration, a general solution is sought that does not require rule-based feature engineering.

Thus, we propose an adaptive weighting method, where we quantitatively measure the reliability (or, rather, lack thereof) of each model $\theta_{i,j}$ by calculating the level of disagreement between $\theta_{i,j}$ and the ensemble, denoted as $a_{i,j,t}$, and then eliminate some of the most unreliable models from the ensemble. The disagreement, $a_{i,j,t}$, can be calculated as the averaged Kullback-Leibler divergence between ensemble predictions and predictions of $\theta_{i,j}$, which can be written as:

$$\begin{aligned} a_{i,j,t} &= \frac{1}{t-1}\sum_{t'=1}^{t-1} D_{\mathrm{KL}}(\bar{\boldsymbol{y}}_{i,t'}||\hat{\boldsymbol{y}}_{i,j,t'}) \\ &= \frac{1}{t-1} D_{\mathrm{KL}}(\bar{\boldsymbol{y}}_{i,t-1}||\hat{\boldsymbol{y}}_{i,j,t-1}) + \frac{t-2}{t-1} a_{i,j,t-1} \end{aligned} \tag{5}$$

where $D_{\mathrm{KL}}(\cdot||\cdot)$ is the Kullback–Leibler divergence.

A higher level of disagreement suggests misbehavior, as what might be expected, for example, when part replacements are introduced, causing mismatches between sensing hardware and downstream AI models. Thus, we only retain the top $K$ models with the lowest $a_{i,j,t}$ in our ensemble and assign equal weights to them, which can be written as:

$$w_{i,j,t} = \begin{cases} 1, & \text{if } a_{i,j,t} \in \min_K \left(\{a_{i,j,t}\}_{i=1,j=1}^{N,M}\right), \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

where $\min_K(\cdot)$ denotes the set of $K$ minimal values in a given set, $\{a_{i,j,t}\}_{i=1,j=1}^{N,M}$ denotes all $a_{i,j,t}$ at time $t$. Our proposed method can be seen as a generalization to the all-equal weighting policy and can be reduced to all-equal weighting when $K = |\{a_{i,j,t}\}_{i=1,j=1}^{N,M}|$.

### C. Implementation Notes

We fully implemented the proposed test-time adaptation system with all the above components based on a pub/sub-based middleware [37]. In particular, for the vehicle classification task, we design each node as a Raspberry Pi 4B+, equipped with a geophone for seismic sensing, a microphone for acoustic data collection, and a GPS module for node localization and time synchronization. A portable battery bank powers each node.

In the target application scenario, each node runs its own classifier models, independently making inference based on data from its local sensors. Importantly, nodes may produce predictions asynchronously and at varying time intervals. As illustrated in Figure 1, the nodes communicate only predicted labels (minimal-sized messages) to a central broker, significantly reducing communication overhead. The broker then computes an ensemble decision based on available predictions collected within a specified spatiotemporal window. To further enhance robustness, the broker employs a threshold-based approach for the spatiotemporal ensemble window: an ensemble prediction is computed and disseminated back to the participating nodes only if a sufficient number of predictions have been received within that window. This design inherently accounts for communication constraints, varying prediction rates, and potential message losses, making the system resilient to intermittent connectivity commonly encountered in IoT deployments.

The implementation comprises approximately 9,000 lines of Python code for the high-level logic, data processing, and neural network components. Additionally, we wrote around 1,300 lines of Rust code for performance-critical sections, particularly those involving low-level system interactions and sensor processes.

TABLE I
Deployment environment characteristics.
Vehicle abbreviations: C (CX-30), G (GLE-350), M (Mustang), X (MX-5)

| ID | Terrain | Weather | Targets | # Nodes | Length |
|---|---|---|---|---|---|
| H08 | Asphalt & gravel | Sunny | C, G, M, X | 6 | 2.77 h |
| H24 | Asphalt & gravel | Rainy | C, G, M, X | 6 | 3.43 h |
| S31 | Dirt & gravel | Sunny | C, G, M, X | 6 | 2.78 h |
| A06 | Asphalt | Sunny | C, G, X | 6 | 2.14 h |
| I29 | Concrete | Windy | C, G, M, X | 8 | 4.14 h |
| I22 | Concrete | Sunny | C, M, X | 8 | 3.00 h |

## V. Evaluation

To evaluate performance, we tested RestoreML in three controlled experimental conditions. Two involve a multi-node vehicle classification system based on acoustic and seismic sensing. The system recognizes the makes and models of vehicles based on their acoustic and seismic signatures. Each sensor node consists of a battery-powered Raspberry Pi 4 with a microphone and a geophone. Four different vehicle models were driven by the authors and the classification results were recorded. The third condition involves an acoustic event detection task that inputs the audio of human activities captured by eight microphones in an office setting. Twelve activities were performed (e.g., "eat", "meet", "enter", "sit", etc.) that are not mutually exclusive. The output is one or more labels, depending on the current activities performed. In all cases, the RestoreML broker runs on an edge server, while classifiers run on local sensing nodes. The server aggregates individual node classification results into ensemble predictions, which are returned to the nodes for pairing with sensor data during TTA. The overall data flow is consistent with Figure 1. Below, we describe these experiments and their results.

### A. Vehicular Experiments

Table I shows the environments in which the vehicle classification experiments were performed. To illustrate the independence of our TTA approach from the specific neural

TABLE II
Performance comparison of applying different TTA methods on target domains S31 and H08. DS-S, DS-A, and Ens. denote the performance of DS-S, DS-A, and the ensemble models. *Worst* is the worst performance of the node (the largest domain shift). *Avg.* is the average performance of the models. Number of parameters: DS-S (0.33M), DS-A (0.33M).

| Method | **S31** | | | | | **H08** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | DS-S | | DS-A | | Ens. | DS-S | | DS-A | | Ens. |
| | Worst | Avg. | Worst | Avg. | | Worst | Avg. | Worst | Avg. | |
| NoAdapt | 9.9 | 10.2 | 83.8 | 84.8 | 89.7 | 43.6 | 49.1 | 77.1 | 81.6 | 90.5 |
| Oracle | 41.7 | 44.8 | 91.9 | 92.6 | 95.6 | 55.1 | 61.1 | 78.4 | 84.8 | 94.7 |
| RestoreML | **41.2** | **45.4** | **89.9** | **91.0** | **94.4** | **55.1** | **59.2** | **78.0** | **84.1** | **91.8** |
| RestoreML-Last+BN | 18.9 | 24.4 | 85.7 | 86.9 | 91.0 | 46.5 | 55.1 | 77.8 | 82.6 | 91.1 |
| RestoreML-Last | 12.1 | 19.5 | 85.4 | 86.1 | 90.5 | 44.8 | 52.4 | 77.5 | 82.4 | 90.9 |
| RestoreML-BN | 16.5 | 23.0 | 85.5 | 86.4 | 90.8 | 46.0 | 55.0 | 77.6 | 82.5 | 91.0 |
| Adaptive-BN | 9.9 | 10.4 | 84.1 | 85.0 | 89.9 | 42.8 | 48.4 | 77.2 | 81.7 | 90.6 |
| TENT | 9.9 | 11.5 | 83.3 | 84.8 | 90.8 | 34.6 | 41.2 | 70.3 | 76.9 | 90.0 |
| TENT-BN | 10.6 | 12.0 | 85.1 | 85.6 | 90.6 | 41.9 | 47.1 | 76.8 | 81.8 | 90.2 |
| Pseudo-label | 10.2 | 11.9 | 83.1 | 84.6 | 90.6 | 36.1 | 41.3 | 73.3 | 78.3 | 89.3 |
| MEMO | 9.7 | 9.8 | 83.0 | 83.8 | 88.8 | 10.9 | 21.0 | 70.1 | 76.0 | 87.6 |
| CoTTA | 9.8 | 10.1 | 41.8 | 44.2 | 42.7 | 24.7 | 30.7 | 53.1 | 61.7 | 64.3 |

TABLE III
Performance comparison of applying different TTA methods on target domain I22, I29, and A06. VibroFM and Ensemble denote the performance of VibroFM, and the ensemble models. *Worst* denotes the worst performance among the models, typically acquired from the node with the worst domain shift. *Avg.* denotes the average performance of the models. Number of parameters: VibroFM (11.78M).

| Method | **I22** | | | | **I29** | | | | **A06** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VibroFM | | Ensemble | | VibroFM | | Ensemble | | VibroFM | | Ensemble | |
| | Worst | Avg. | Worst | Avg. | Worst | Avg. | Worst | Avg. | Worst | Avg. | Worst | Avg. |
| NoAdapt | 61.3 | 72.6 | **69.0** | 79.3 | 40.6 | 72.8 | **65.4** | 84.0 | 67.2 | 71.3 | **80.8** | 85.0 |
| Oracle | 69.4 | 83.6 | 78.1 | 89.7 | 40.6 | 79.6 | 68.5 | 89.1 | 74.2 | 82.4 | 86.6 | 91.3 |
| RestoreML | **64.9** | **79.3** | 67.5 | **83.8** | **40.6** | **77.0** | 64.8 | **84.9** | **70.6** | **80.2** | 76.9 | **87.4** |
| TENT | 26.9 | 55.4 | 42.5 | 57.5 | 22.4 | 67.1 | 50.5 | 74.2 | 67.9 | 76.9 | 79.0 | 86.7 |
| Pseudo-label | 45.6 | 67.3 | 63.0 | 72.5 | 40.6 | 72.2 | 62.6 | 78.0 | 67.6 | 75.9 | 78.5 | 86.0 |

network architecture, the sensing modalities, and the initial neural network model training approach used, we adopt three different neural network models for vehicle classification (to be adapted with TTA): (1) DS-A: a DeepSense [38] model initially trained on *labeled acoustic* signals; (2) DS-S: a DeepSense model initially trained on *labeled seismic* signals; and (3) VibroFM [39]: a large Transformer [40] based IoT foundation model that is first *pre-trained on a large-scale unlabeled dataset* using self-supervised learning and then *fine-tuned* on a labeled dataset, using both acoustic and seismic signals. All three models take 2-second data windows as input and output a single (nearest) target class per window.

To experiment with domain shift in IoT deployment, we train the neural network model using data from one environment and then explore the success of our test-time adaptation algorithm in adapting the model to another environment.

We compare our method with seven TTA baselines, namely: (1) *No adaptation*: a trivial baseline where no domain adaptation is performed; (2) *Adaptive-BN* [20]: dynamically adjusting batch normalization layer parameters on target domain; (3) *TENT-all* [6]: minimizing the entropy of prediction, tuning all layers; (4) *TENT-BN* [6]: minimizing entropy of prediction, tuning only batch normalization layers; (5) *Pseudo-label* [21]: fine-tine the model with the class with highest probability; (6) *MEMO* [8]: minimizing entropy with data augmentation; and (7) *CoTTA* [13]: fine-tune with a moving-averaged teacher with dynamic parameter reset.

Apart from the baselines, we also include three ablations of RestoreML in the comparison to evaluate our method when only part of the model can be trained due to computational power constraints. They are (1) *RestoreML-Last*: where only the last layer of the models is fine-tuned using TTA; (2) *RestoreML-BN*: where only the batch normalization layer of the models is fine-tuned using TTA; and (3) *RestoreML-Last+BN*: where only the last layer and batch normalization layers of the models are fine-tuned using TTA. We also include an *Oracle* method to show an **upper bound** of TTA performance, where *ground-truth labels* for the target domain data are used to fine-tune the models. Since we focus on measuring the performance of the classification task, we choose the commonly used *Macro-F1 score* throughout this paper.

*1) Adapting Supervised Models Outdoors:* In the first experiment, we use our algorithm to adapt two classifiers trained in a supervised manner, namely, an acoustic classifier DS-A and a seismic classifier DS-S. In this experiment, we train DS-S (0.33M parameters) and DS-A (0.33M parameters) models on the H24 environment in a supervised way, which serves as the source domain (please refer to Table I for a description of H24). Then, we test these trained models in the H08 and S31 environments. The former differs from the source domain in weather conditions, whereas the latter differs in terrain type (see Table I for details). Sensors in the target environments were deployed close together. Vehicles of four classes were driven past these sensors one at a time. We set the spatial ensemble window to include all nodes and the temporal ensemble window to two seconds. The results are shown in Table II. The DS-S model achieves relatively lower classification performance compared to DS-A, including under Oracle conditions. This lower performance is primarily due to the limited discriminative power of seismic signals alone in this deployment. However, despite the lower standalone accuracy, incorporating DS-S sensors into the classification system remains beneficial. Specifically, seismic sensors are significantly less affected by environmental disturbances such as wind noise and background acoustic interference, thus providing complementary information that can enhance overall system robustness. Additionally, the notable performance discrepancy observed between stations S31 and H08 is attributable to differences in local environmental conditions and background noise levels affecting each sensor site differently. In both cases, however, the overall results reveal that RestoreML consistently achieves the best performance on all metrics in both target domains among all TTA techniques. RestoreML's performance surpasses baselines by a large margin and closely approaches the *Oracle* method. Besides, RestoreML with only the last layer or batch normalization layers fine-tuned also exhibit superior performance compared to baselines, showing RestoreML's strong capability on resource-constrained platforms.

*2) Adapting Self-Supervised Models Outdoors:* Next, we explore TTA in the context of a self-supervised model. In this experiment, we pre-train and fine-tune a self-supervised AI model, VibroFM (11.78M parameters), using data from all three environments mentioned above, H08, H24, and S31, and then test it in three new target environments: (1) I22, (2) I29, (3) A06 (see Table I for descriptions). Three notable aspects make this experiment more challenging than the previous one: (1) The I22, I29, and A06 sensor deployments are more spread out (i.e., the sensors are further apart). Thus, different sensor nodes do not observe the same target(s) simultaneously, which increases the challenge of producing accurate ensemble predictions. (2) We allowed up to two vehicles to move simultaneously, such that ground truth labels could be different for different nodes. (3) I22 and I29 feature new nodes joining the system after deployment (i.e., hardware not used in collecting training data). They represent newly restored nodes with a different device but are still running the old model, which may increase the difficulty of TTA. Due to the larger scale of the target deployments in this experiment, we choose a temporal ensemble window of 10 seconds and a separate spatial ensemble window for each node that includes its direct neighbors only. We exclude the baselines Adaptive-BN and TENT-BN that rely on batch normalization layers, since those layers are not present in the VibroFM model used in this experiment, rendering those baselines inapplicable. We also exclude MEMO and CoTTA due to their poor performance relative to other baseline, as seen in Table II. Hence, they warrant no further consideration.

Table III presents the results of this experiment. As observed, RestoreML achieved the highest F1 on most metrics compared to baselines, showcasing its versatility on large pre-trained foundation models.

TABLE IV
Classification F1 (%) on MM Office. Number of parameters: BEATs (90.35M), DS (0.33M)

| Method | BEATs | | DS | |
|---|---|---|---|---|
| | WN | Avg. | WN | Avg. |
| NoAdapt | 67.4 | 72.9 | 74.9 | 81.2 |
| Oracle | 78.2 | 78.7 | 99.1 | 99.4 |
| RestoreML | **74.4** | **75.3** | **87.8** | **88.2** |
| TENT | 36.1 | 35.6 | 47.5 | 52.0 |
| Pseudo-label | 36.8 | 43.0 | 47.5 | 48.3 |

## *B. Adapting Indoors*

To demonstrate the versatility of RestoreML, we additionally evaluate it in an acoustic event detection task in an office environment [41]. Two AI models are used for this task. (1) BEATs [42] (90.35M parameters), a state-of-the-art self-supervised learning framework for audio representation pre-training. We use the BEATs model pre-trained[2] on AudioSet-2M as the backbone, and fine-tune for linear classification. (2) DeepSense [38] (0.33M parameters), a CNN-RNN-based network. We trained an end-to-end classifier using DeepSense. Four nodes are used at test time, each assigned to process data from one of the microphones in the *right* room. The models on these nodes share identical initial weights, fine-tuned using data from the *left* room. All nodes perform inference and adaptation collaboratively using RestoreML.

The classification F1 scores of RestoreML and baselines (TENT and Pseudo-label) are shown in Table IV. Note that DeepSense is an end-to-end trained, much smaller network than the pre-trained BEATs model and allows fine-tuning of the entire network. This ability to fine-tune the full model enables DeepSense to perform better adaptation. RestoreML outperforms the baselines across tasks, highlighting its versatility and effectiveness in various settings and with different neural networks.

To illustrate model performance improvements during TTA, Figure 2 illustrates how RestoreML adapts models of different initial quality. In this run, initially, three out of four nodes ($n0$, $n1$, and $n2$) start out relatively well-tuned to the deployment environment, while one node, $n3$, experiences degradation. As shown in the curve, $n3$'s classification F1 score improves by approximately 10% over time, eventually catching up with the other nodes without negatively impacting their performance.

Evaluation results presented above confirm that our proposed TTA solution offers a practical means for fine-tuning

[2]https://github.com/microsoft/unilm/tree/master/beats

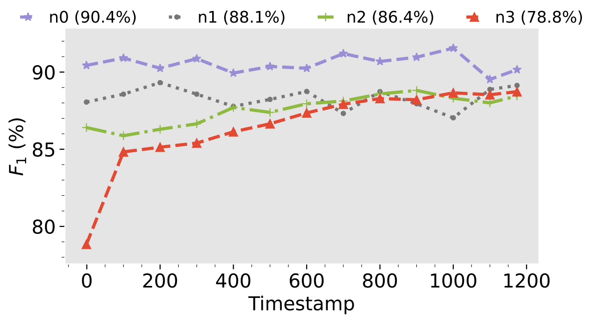

Fig. 2. Illustration of model adaptation over time. A newly introduced, less-adapted node (n3) improves over time by learning from well-adapted, older nodes. The node legend is ordered based on initial performance.

IoT nodes in the field in an *unsupervised* manner, especially in situations calling for fine-tuning of replacement nodes (i.e., where only a minority of nodes need to be adapted at any given time). Initial evidence suggests that the solution can work with different AI models and results in performance improvement across multiple applications and modalities. Our released dataset will hopefully facilitate further research on the topic.

## VI. Conclusion

This paper introduced RestoreML, a test-time adaptation technique that facilitates model fine-tuning post-deployment without requiring labeled data. We proposed a least-trained-balanced sampling policy and a dynamic weights determination algorithm to overcome TTA challenges presented by IoT monitoring systems, while exploiting IoT deployment characteristics. Our comparison with the baselines and ablation studies validate the superior performance of RestoreML. While this paper offers a proof of concept for the success of TTA in IoT settings, further exploration would be beneficial to fully uncover the potential of RestoreML and improve its behavior. A more systematic exploration is needed of the effects of deployment properties (e.g., area covered by sensor network), background noise characteristics, number of concurrent items or activities to classify, AI model and model size used, size of training data, and the duration of TTA. Since TTA requires local storage, effective policies are also needed for data replacement/eviction when local storage limits are reached. These topics constitute rich opportunities for follow-up research and are delegated to future work.

## Acknowledgements

Research reported in this paper was sponsored in part by DEVCOM ARL under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA), and in part by NSF CNS 20-38817, and the Boeing Company. It was also supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The views and conclusions contained in this document are those of the authors, not the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

## References

[1] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, “Adapting visual category models to new domains,” in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer, 2010, pp. 213–226.
[2] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa, “Visual domain adaptation: A survey of recent advances,” *IEEE signal processing magazine*, vol. 32, no. 3, pp. 53–69, 2015.
[3] X. Liu, C. Yoo, F. Xing, H. Oh, G. El Fakhri, J.-W. Kang, J. Woo *et al.*, “Deep unsupervised domain adaptation: A review of recent advances and perspectives,” *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.
[4] C. Fang, S. Liu, Z. Zhou, B. Guo, J. Tang, K. Ma, and Z. Yu, “Adashadow: Responsive test-time model adaptation in non-stationary mobile environments,” in *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, 2024, pp. 295–308.
[5] J. Liang, R. He, and T. Tan, “A comprehensive survey on test-time adaptation under distribution shifts,” *International Journal of Computer Vision*, pp. 1–34, 2024.
[6] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell, “Tent: Fully test-time adaptation by entropy minimization,” *arXiv preprint arXiv:2006.10726*, 2020.
[7] Z. Nado, S. Padhy, D. Sculley, A. D’Amour, B. Lakshminarayanan, and J. Snoek, “Evaluating prediction-time batch normalization for robustness under covariate shift,” *arXiv preprint arXiv:2006.10963*, 2020.
[8] M. Zhang, S. Levine, and C. Finn, “Memo: Test time robustness via adaptation and augmentation,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 38 629–38 642, 2022.
[9] Q. Guo, X. Wang, Y. Wu, Z. Yu, D. Liang, X. Hu, and P. Luo, “Online knowledge distillation via collaborative learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 020–11 029.
[10] D. Sahoo, Q. Pham, J. Lu, and S. C. Hoi, “Online deep learning: Learning deep neural networks on the fly,” *arXiv preprint arXiv:1711.03705*, 2017.
[11] A. Bobu, E. Tzeng, J. Hoffman, and T. Darrell, “Adapting to continuously shifting domains,” 2018.
[12] M. Wulfmeier, A. Bewley, and I. Posner, “Incremental adversarial domain adaptation for continually changing environments,” in *2018 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4489–4495.
[13] Q. Wang, O. Fink, L. Van Gool, and D. Dai, “Continual test-time domain adaptation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7201–7211.
[14] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.
[15] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
[16] R. Li, Q. Jiao, W. Cao, H.-S. Wong, and S. Wu, “Model adaptation: Unsupervised domain adaptation without source data,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9641–9650.
[17] H.-W. Yeh, B. Yang, P. C. Yuen, and T. Harada, “Sofa: Source-data-free feature alignment for unsupervised domain adaptation,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 474–483.
[18] S. Goyal, M. Sun, A. Raghunathan, and J. Z. Kolter, “Test time adaptation via conjugate pseudo-labels,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 6204–6218, 2022.
[19] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, “Revisiting batch normalization for practical domain adaptation,” *arXiv preprint arXiv:1603.04779*, 2016.
[20] S. Schneider, E. Rusak, L. Eck, O. Bringmann, W. Brendel, and M. Bethge, “Improving robustness against common corruptions by covariate shift adaptation,” *Advances in neural information processing systems*, vol. 33, pp. 11 539–11 551, 2020.
[21] D.-H. Lee *et al.*, “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks,” in *Workshop on challenges in representation learning, ICML*, vol. 3, no. 2. Atlanta, 2013, p. 896.

[22] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4320–4328.

[23] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," *arXiv preprint arXiv:1804.03235*, 2018.

[24] X. Zhu, S. Gong *et al.*, "Knowledge distillation by on-the-fly native ensemble," *Advances in neural information processing systems*, vol. 31, 2018.

[25] D. Chen, J.-P. Mei, C. Wang, Y. Feng, and C. Chen, "Online knowledge distillation with diverse peers," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 3430–3437.

[26] J. Li, S. Wu, C. Liu, Z. Yu, and H.-S. Wong, "Semi-supervised deep coupled ensemble learning with classification landmark exploration," *IEEE Transactions on Image Processing*, vol. 29, pp. 538–550, 2019.

[27] G. Wu and S. Gong, "Peer collaborative learning for online knowledge distillation," in *Proceedings of the AAAI Conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 10 302–10 310.

[28] L. Li and Z. Jin, "Shadow knowledge distillation: Bridging offline and online knowledge transfer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 635–649, 2022.

[29] L. Zhang, D. Zheng, M. Yuan, F. Han, Z. Wu, M. Liu, and X.-Y. Li, "Multisense: Cross-labelling and learning human activities using multimodal sensing data," *ACM Transactions on Sensor Networks*, vol. 19, no. 3, pp. 1–26, 2023.

[30] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[31] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[32] C. Wang, G. Yang, G. Papanastasiou, H. Zhang, J. J. Rodrigues, and V. H. C. De Albuquerque, "Industrial cyber-physical systems-based cloud iot edge for federated heterogeneous distillation," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5511–5521, 2020.

[33] P. Qi, X. Zhou, Y. Ding, Z. Zhang, S. Zheng, and Z. Li, "Fedbkd: Heterogenous federated learning via bidirectional knowledge distillation for modulation classification in iot-edge system," *IEEE Journal of Selected Topics in Signal Processing*, vol. 17, no. 1, pp. 189–204, 2022.

[34] K. Ozkara, N. Singh, D. Data, and S. Diggavi, "QuPeD: Quantized Personalization via Distillation with Applications to Federated Learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 3622–3634. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/1dba3025b159cd9354da65e2d0436a31-Abstract.html

[35] A. Mohammed and R. Kora, "A comprehensive review on ensemble deep learning: Opportunities and challenges," *Journal of King Saud University-Computer and Information Sciences*, vol. 35, no. 2, pp. 757–774, 2023.

[36] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, "Ensemble deep learning: A review," *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105151, 2022.

[37] J. Li, Y. Chen, T. Kimura, T. Wang, R. Wang, D. Kara, Y. Hu, L. Wu, W. A. Hanafy, A. Souza, P. Shenoy, M. Wigness, J. Bhattacharyya, J. Kim, G. Wang, G. Kimberly, J. Eckhardt, D. Osipychev, and T. Abdelzaher, "Acies-OS: A content-centric platform for edge AI twinning and orchestration," in *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*. Big Island, HI, 2024, pp. 1–1.

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

[39] T. Kimura, J. Li, T. Wang, D. Kara, Y. Chen, Y. Hu, R. Wang, M. Wigness, S. Liu, M. Srivastava *et al.*, "On the efficiency and robustness of vibration-based foundation models for iot sensing: A case study," *arXiv preprint arXiv:2404.02461*, 2024.

[40] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[41] M. Yasuda, Y. Ohishi, S. Saito, and N. Harado, "Multi-View And Multi-Modal Event Detection Utilizing Transformer-Based Multi-Sensor Fusion," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 4638–4642.

[42] S. Chen, Y. Wu, C. Wang, S. Liu, D. Tompkins, Z. Chen, W. Che, X. Yu, and F. Wei, "BEATs: Audio Pre-Training with Acoustic Tokenizers," in *Proceedings of the 40th International Conference on Machine Learning*. PMLR, Jul. 2023, pp. 5178–5193. [Online]. Available: https://proceedings.mlr.press/v202/chen23ag.html