Few-Shot Adaptation to Unseen Conditions for Wireless-based Human Activity Recognition without Fine-tuning

Xiaotong Zhang*, *Student Member, IEEE,* Qingqiao Hu*, *Student Member, IEEE,* Zhen Xiao, *Student Member, IEEE,* Tao Sun, *Student Member, IEEE,* Jiaxi Zhang, *Student Member, IEEE,* Jin Zhang, *Member, IEEE,* and Zhenjiang Li, *Member, IEEE*

Abstract—Wireless-based human activity recognition (WHAR) enables various promising applications. However, since WHAR is sensitive to changes in sensing conditions (e.g., different environments, users, and new activities), trained models often do not work well under new conditions. Recent research uses meta-learning to adapt models. However, they must fine-tune the model, which greatly hinders the widespread adoption of WHAR in practice because model fine-tuning is difficult to automate and requires deep-learning expertise. The fundamental reason for model fine-tuning in existing works is because their goal is to find the mapping relationship between data samples and corresponding activity labels. Since this mapping reflects the intrinsic properties of data in the perceptual scene, it is naturally related to the conditions under which the activity is sensed. To address this problem, we exploit the principle that under the same sensing condition, data of the same activity class are more similar (in a certain latent space) than data of other classes, and this property holds invariant across different conditions. Our main observation is that meta-learning can actually also transform WHAR design into a learning problem that is always under similar conditions, thus decoupling the dependence on sensing conditions. With this capability, general and accurate WHAR can be achieved, avoiding model fine-tuning. In this paper, we implement this idea through two innovative designs in a system called RoMF. Extensive experiments using FMCW, Wi-Fi and acoustic three sensing signals show that it can achieve up to 95.3% accuracy in unseen conditions, including new environments, users and activity classes.

Index Terms—Mobile computing, activity recognition, few-shot learning, wireless sensing.

1 Introduction

Wireless-based Human Activity Recognition (WHAR) uses wireless signals as the sensing media to recognize human activities or gestures [1], [2]. Due to the advantages of through-wall sensing, robustness to ambient light conditions, and better protection of user privacy than other peer technologies such as computer vision [3], [4], the WHAR technology can enable many promising and useful applications in human-computer interaction [5], healthcare [6], [7], smart home [8], [9], etc. With the recent success in these applications, an emerging and important topic is how to maintain good performance when deploying WHAR systems in **unseen conditions** after training [10], which includes 1) new users with different behavior habits, 2) new environments with different room layouts and locations of transceivers and 3) even new classes of activities or gestures never appeared in the training process.

Recent research finds that an effective solution to the unseen condition is to employ meta-learning [10], leveraging its powerful capability for condition generalization. The main idea is to organize the training dataset into groups, called *tasks*. Each task contains only a small number of data samples from several activities of a subset of users and environments in the training dataset. Therefore, the sensing conditions involved in different tasks are different. The data samples of each task are further divided into two sets: the **support set** and the **query set**. We can then train the WHAR model on a task's support set and validate it on its query set. After one task is trained, the data samples of the other task are from a new condition to the current model. Therefore, different tasks simulate different conditions, and training essentially rehearses how to use few samples from a certain condition (in the support set) to update the model so that it can perform well under this condition (in the corresponding query set). With this capability, for any new condition after training, people only need to collect few labeled data samples (shots) from it to fine-tune the model, and the model can quickly adapt to the condition. However, no matter how the number of data shots is reduced [11], the model always needs to be fine-tuned, which greatly hinders the widespread adoption of WHAR in practice, since fine-tuning the model is difficult to automate, requires expertise in deep learning, and demands several times the hardware

* *Both authors contributed equally to this research.*

- *X. Zhang is with the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China and the Department of Computer Science, City University of Hong Kong, Hong Kong, China. E-mail: xzhang2587-c@my.cityu.edu.hk.*
- *Q. Hu is with the Department of Computer Science, Stony Brook University, New York, U.S. and the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. E-mail: qingqiao.hu@stonybrook.edu.*
- *T. Sun is with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. E-mail: 12232426@mail.sustech.edu.cn.*
- *Jiaxi Zhang is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China and the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. E-mail: jzhanghl@connect.ust.hk.*
- *Jin Zhang is with the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China and Peng Cheng Laboratory, Shenzhen, China. E-mail: zhangj4@sustech.edu.cn.*
- *Z. Xiao and Z. Li are with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. E-mail: zxian4@cityu.edu.hk, zhenjiang.li@cityu.edu.hk.*
- *The corresponding authors are Zhenjiang Li and Jin Zhang.*

resources compared to the inference process.

To push the design of WHAR systems a step further, we wonder: *is it possible to quickly adapt to unseen conditions without model fine-tuning?* To this end, we first delve into why fine-tuning cannot be removed from existing methods. Existing WHAR systems introduce advanced models to explore and characterize complex mapping relationships from data samples to corresponding labels. They are good at mining the internal features of data samples to discover such relationships, but data samples from different conditions follow different feature distributions, resulting in distinct data-to-label mappings. Thus, under unseen conditions, the previously learned mapping relationship will be invalid, and the model has to be fine-tuned to correct it. Although meta-learning can effectively reduce the adaption overhead, the fine-tuning process is always required in existing methods.

To avoid model fine-tuning, we revisit the model design from a different angle. The data samples of the support set and the query set of each task come from the same sensing condition and have similar feature distributions. Therefore, these data samples of the same activity class are more similar to each other in a certain latent space than data samples of other classes. Since WHAR is essentially a classification problem, with this observation, we can come up with suitable metrics to directly measure the similarity of data samples and design a clustering-like model — for each sample in the query set, we check which data sample in the support set has the highest similarity and use its label as the classification result. After training, we also collect few labeled data samples for any new unseen condition in real use, but we only use them to form a **reference set** (similar to the support set in training), not for model fine-tuning. We then treat the actual sensing data as a **test set** (similar to the query set). Data samples in the reference set will input into our model, and through similarity comparison, the model will output the label of the sample in the reference set that is most similar to the sample in the test set as the recognition result.

The advantage of our method is that in unseen conditions, although the data-to-label mapping would change, the data samples from the reference and test sets still come from the same condition. Hence, the above-mentioned similarity observation between data samples can still hold (with our new design), which provides the opportunity to apply the model directly to new conditions without fine-tuning. The main insight of our design is that meta-learning already ensures the data, which the model needs to use in the respective phases of training and testing, come from the same conditions. Our method explicitly exploits this phenomenon in its design. It attempts to directly employ this inherently invariant property of the data to bypass updating the model, *i.e.*, fine-tuning, instead of focusing on the different data-to-label mappings under different conditions, which is essentially a more complex problem, but not necessary for WHAR. We note that our method is not intended to replace the mainstream model design of learning data-to-label mappings, while it just fits better with WHAR to handle unseen conditions.

However, our system cannot be achieved using traditional clustering methods due to two unique challenges. First, how to effectively measure data similarity to distinguish different types of data samples? Metrics used in traditional clustering methods, such as Euclidean distance and cosine distance, cannot robustly distinguish data samples from various wireless sensing signals. We need a new and specialized design to transform the sensing data into a latent space suitable for our design principle, so that an effective and robust data similarity measure can be achieved. Second, how to efficiently support new classes? Adding new classes (activities) to a model often requires changes to the model structure, such as fully connected layers, which necessitates fine-tuning the model. However, we want the system to allow new classes to be added, but still avoid model fine-tuning.

To address these challenges, we propose a design, called RoMF. In RoMF, we formulate the learning of WHAR data similarity as a feature propagation problem and utilize graph neural network (GNN) [12] to find accurate and reliable metrics to evaluate the similarity between wireless signals through novel GNN node construction and updating. We further integrate the GNN model into a meta-learning framework to "teach" the model how to compare. Similar to existing meta-learning, we divide the training data from each task into a support set and a query set with labeled data. By building GNN nodes, we explicitly combine label information with data in the support set, making the model learn a dedicated metric that once two samples have different labels, the distance between them should be large. For the scenario of adding new classes, we further propose a novel decompose-and-vote mechanism for our GNN model, so that new activity classes can be added freely without model fine-tuning.

We develop RoMF with a unified input format to easily work with various wireless sensing signals. We experiment with frequency-modulated continuous-wave (FMCW), Wi-Fi and acoustic sensing signals. The evaluation involves a total of 73 users (40 FMCW users, 17 Wi-Fi users and 16 acoustic users) with 44 daily activities in 27 different environments. We compare RoMF with the state-of-the-art methods RF-Net [11] and OneFi [13] in three scenarios, including 1) new environments only, 2) new environments and users, and 3) new environments, users and classes, where the number of new classes is up to 10. RoMF is trained only using several activity data of a subset of users in two environments, while it can perform activity recognition directly in these unseen condition scenarios without fine-tuning the model and achieve 84.7–95.3% accuracy, outperforming RF-Net and OneFi by 2.2–73.1%. We will open source all RoMF code to facilitate future research in this field. In summary, this paper makes the following contributions:

- We design RoMF to adapt the WHAR system to new conditions without fine-tuning the model. At the same time, it retains good accuracy and few-shot adaptability. The design of RoMF can significantly improve the real-world applicability of this sensing technology.
- We propose a novel GNN-based meta-learning framework to achieve the goal of avoiding model fine-tuning through a novel GNN design and an effective mechanism to support the recognition of

new activities.
- We develop a prototype of RoMF and conduct extensive real-world experiments. The results show the efficacy of our design under various unseen conditions.

2 Background

2.1 Unseen Sensing Conditions

A major obstacle limiting WHAR's widespread use is that the feature distribution of wireless sensing signals can change significantly under various unseen sensing conditions:

- **i) New environments:** Different room layouts and sensing device locations/orientations can lead to differences in sensing signals for the same activity [14], [15]. For example, a room with dense furniture may have more multipath reflections during sensing.
- **ii) New users:** Different behavioral habits and physical characteristics of users also make the wireless signals reflected by the same activity different [16]. For example, users often have different patterns for "walking" in terms of stride length and speed. These differences are difficult to characterize in advance.
- **iii) New activity classes:** New activities not included in the training data set [13] can be added to the existing WHAR system to enrich its functionality.

Existing solution. However, training a new model for each condition is impractical due to the large amount of conditions and the heavy overhead of labeled data collection. Recent research found an effective solution [10], which employs few-shot learning to quickly adapt the WHAR model to new conditions in two steps:

Step-1) Few-shot data collection. For the new condition of deploying the WHAR system after training, we can first collect some sensing data samples from it and annotate them with activity labels. Recent studies have found that very few shots (*e.g.*, 1–3) will be enough with meta-learning [17]. This minimal data collection does not introduce any additional burden, as data collection is a fundamental requirement in any WHAR systems.

Step-2) Model fine-tuning. We then use these few shots of data samples to fine-tune the model for adaptation to new conditions. However, fine-tuning requires deep learning expertise and is difficult to automate. Recent research points out that fine-tuning is a limitation of such designs [18].

Our objective. In this paper, we still follow step-1) using few shots of labeled data collected under the target new sensing condition (to build the reference set), but avoid model fine-tuning in step-2). For WHAR, since the label is just the name of each activity, the collection of a reference set and its labeling can be developed into an easy-to-operate system setup routine, which does not bring much overhead. In RoMF, we focus on avoiding model fine-tuning in the second step to improve the applicability of WHAR in practice.

2.2 Modeling Sensing Signals

RoMF can support a variety of wireless sensing signals, and we take three popular signals of FMCW, Wi-Fi, and acoustic sound as examples to explain its design. For the convenience of development, we unify the input format of different sensing signals into a two-dimensional (2D) matrix I to the system, which can be regarded as a 2D image. Below, we detail the preprocessing of different sensing signals.

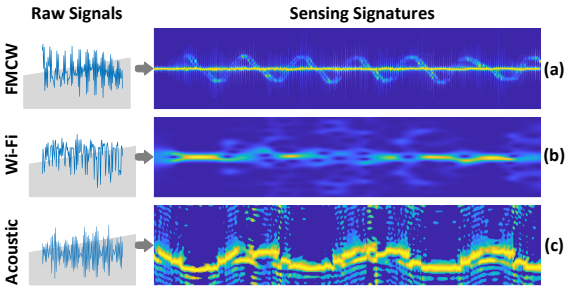


Fig. 1: Illustration of RoMF's input from different sensing signals for a same push-pull activity. As signals characterize activities differently, similar to existing methods, we will train different versions of RoMF with the same model structure but different parameters when using different types of sensing signals.

FMCW. FMCW radars emit a sinusoidal signal called a chirp whose frequency increases linearly in a wide frequency band B from a starting frequency f_0 over a time T [19]. Once the chirp signal is reflected from the sensing target and received by the radar, the radar generates an intermediate frequency (IF) signal of constant frequency equal to the frequency difference of the transmitted and received chirps as follows:

$$y_f(t) \quad = \quad A \times e^{j2\pi(f_0\tau - \frac{B\tau^2}{2T} + \frac{B\tau t}{T})}, \tag{1}$$

where A is the amplitude and τ is the round-trip time delay of the signal between the radar and the sensing target. The IF signal is the signal used by the FMCW radar to measure the distance, velocity and angle of the sensing target, and it is usually also the input source of many FMCW-based WHAR methods [20], [21]. Specifically, the IF signal is transformed into a 2D velocity-time map by short-time Fourier transform (STFT), called micro-Doppler spectrum [22], which is the input matrix of RoMF with FMCW. Figure 1(a) shows an example of this spectrum for a push-pull activity. In RoMF, the dimension of each spectrum is 128 $\times$ 512 for FMCW.

Wi-Fi. For Wi-Fi, channel state information (CSI) is widely used for wireless sensing [23]. Denoting the number of subcarriers as C, the CSI component in the c-th subcarrier is:

$$y_w(f_c, t) \quad = \quad \sum_{k=1}^{K} A_k(t) \times e^{-j2\pi f_c \tau_k t}, \tag{2}$$

where K is the number of multipaths, A_k is the amplitude and τ_k is the propagation delay. Both the original CSI and other variants (such as micro-Doppler spectrum) are used in existing Wi-Fi based WHAR designs [24], [25], [26]. Similar to FMCW, we also convert CSI to micro-Doppler spectrum in RoMF with dimension 121 $\times$ 512. Figure 1(b) shows an example spectrum from Wi-Fi for the same activity.

Acoustic sounds. In addition to radio frequency signals, acoustic sounds are also commonly used for sensing and

recognition [27], [28], since mobile devices can easily generate sounds that are inaudible to the human ear (without disturbing us) through device speakers [29]. Sounds can be generated in different forms, and we use a continuous wave signal with a constant frequency f [30] (*e.g.*, $f = 17$ KHz), which can be expressed as $r(t) = 2Acos(2\pi ft - 2\pi 2d/v - \theta_p)$, where v is the sound speed, θ_p is the initial phase, and d the distance from the device to the target. The received reflected signal is firstly multiplied by $cos(2\pi ft)$ and $-sin(2\pi ft)$ respectively to generate the sensing signal [30]:

$$y_a(t) \quad = \quad A \times e^{-j(4\pi fd/v + \theta_p)}, \tag{3}$$

where A is the amplitude. For reliable sensing, we transmit acoustic signals at multiple frequencies, separated by a frequency interval (*e.g.*, 350 Hz), and then perform an Inverse Discrete Fourier Transform (IDFT) for Eq. (3) to obtain the distance spectrum as model input, which is 128 $\times$ 512 in RoMF. Figure 1(c) shows an example for the same activity.

2.3 Meta-learning Basics

One of human intelligence is to learn from existing tasks and quickly understand new tasks with only few labeled observations. For example, knowing the characteristics of a horse can help to learn the characteristics of other related animals such as zebras. Furthermore, even if the child did not know the precise definition or characteristics of a zebra, after viewing several pictures of zebras, the child was able to pick out pictures of zebras from other animals such as dogs, cats, and elephants. Meta-learning aims to imitate such human intelligence, and its key ideas are twofold: 1) learn knowledge from environments (similar to different sensing conditions in WHAR) in training datasets with rich data, and 2) apply accumulated knowledge to new environments for quick adaptation with few labeled samples.

Meta-learning organizes the training process into a sequence of *tasks* and focuses on classifying activities under the same condition each time. For each task, we select only a small amount of data belonging to a subset of all sensing conditions (including environment, user and activity class) in the training dataset, and the selected data and its conditions are different for each task. Assuming we choose four samples for each of five activities in the task, we split them into **a support set** (three samples per activity) and **a query set** (one sample per activity). Since all these data samples are labeled, meta-learning will train the model on the support set, test performance on the query set, and use the test results to update the model. For the next task, the combination of activities, environments, and users will be different.

The rationale behind this arrangement is to rehearse the few-shot adaptation under unseen conditions, *i.e.*, each task is a new condition for the other tasks. The data in the support set simulates data samples collected from a real new condition in the future deployment (*i.e.*, **the reference set**), while the data in the query set simulates the actual sensing data from this new condition for recognition (*i.e.*, **the test set**). By repeating this process, the training aims to optimize the model parameters θ by:

$$\theta^{\star} \quad = \quad \arg\min_{\theta} \sum\nolimits_{\mathcal{T}_i} \mathcal{L}(\mathcal{D}_{\mathcal{T}_i}; \theta), \tag{4}$$

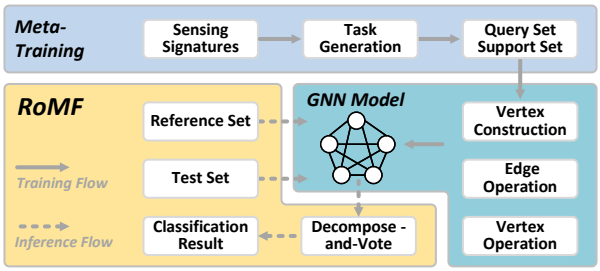


Fig. 2: Overview of the RoMF architecture.

where $\mathcal{L}$ is the loss function and $\mathcal{D}_{\mathcal{T}_i}$ is the data of the support set and the query set for task $\mathcal{T}_i$. In RoMF, we also employ meta-learning to preserve this adaptability, so that few-shot reference data samples can directly guide our proposed model for correct recognition without model fine-tuning.

3 System Design

Figure 2 shows an overview of the RoMF design. We customize a meta-learning framework to train our GNN based recognition model. The meta-learning takes the 2D sensing signatures (§2.2) converted from sensing signals in the training dataset as input, and further divides them into a support set and a query set for each task (§3.3). RoMF model contains efficient elements and operations (§3.1) to learn data similarity. It also exploits a decompose-and-vote block (§3.2) to manage the model structure to support adding new activities for actual deployment. When RoMF is deployed after training, few shots of labeled reference data samples are collected during setup, which will be used as model input. Then, RoMF can accept the sensing signal and directly perform activity recognition with high accuracy (§3.3). In the following, we describe the design of each module.

3.1 GNN-based Recognition Model

In RoMF, our goal is to recognize activities by finding reliable distance measures and comparing similarities between data samples. To mine data similarity, graph neural network (GNN) is an advanced method in the field of machine learning [31], built on the concept of *vertices* and *edges*. Vertices can be designed to encode information related to data samples, *e.g.*, to express data features, and edges are associated with weights that measure the similarity of two vertices connected by the edge. Unlike traditional clustering, vertices can be carefully constructed for better feature representation, and the features of two vertices can be compared by an advanced neural network to compute weights to explore better metrics for more reliable similarity measurements. Therefore, we design our recognition model in RoMF using GNN. In §4.2, our experiments show that using GNN can lead to ∼40% improvements than using traditional clustering algorithms. However, GNN is just a framework, and we need to instantiate every component and operation for WHAR.

Figure 3 shows the high-level idea of our GNN model design. We first construct vertices for each data sample i, where the number of vertices is equal to the total number of data samples in support set and query set in each meta-learning task (§3.3). Then we introduce two operations, edge

operation and vertex operation, to update the weight of each edge and the content of each vertex during training, so that vertices corresponding to the same class are measured closer to each other (otherwise they will be farther away from each other). Finally, for the convergent vertex of the data sample in the query set, we use the label of the most similar sample in the support set as its recognition result, and refer to the original label in the query set of this data sample to minimize a cross entropy loss, thus realizing supervised training (§3.3). Below we introduce our vertex construction design, and the edge and vertex operations.

3.1.1 Vertex construction

A graph can be expressed as $G = (V, E)$, where V is its vertices and E is a matrix consisting all edges with the size of $|V| \times |V|$. We first construct all the vertices $v_i \in V$ for the GNN model. As mentioned above, each vertex is intended to accommodate the features of the corresponding data sample. For each sample in the support set, in addition to the features extracted from the data sample itself, we propose to also encode its label information into the vertices as part of the overall feature representation, since support set samples will act as anchors for query set samples to compare with. Thus, the label information encoded in vertices can serve as a strong indicator to teach the model how to update the content of each vertex during training, making vertices belonging to the same class more similar. Therefore, we use the sensing signature x_i of the data sample i and its label y_i to construct a (type-one) vertex v_i:

$$\text{Type-one vertex:} \qquad v_i =< r(x_i),\ o(y_i) >, \tag{5}$$

where x_i is processed by a small ResNet (only ten layers for lightweight development), denoted as $r(\cdot)$, to extract preliminary data features, and the label y_i is processed by the one-hot encoding processing function $o(\cdot)$ to represent class variables as numeric values [32]. Then, $r(x_i)$ and $o(y_i)$ are concatenated to form a vertex in the form of an one-dimensional vector $< r(x_i),\ o(y_i) >$, as shown in Figure 3.

However, type-one vertex cannot be used to construct vertices for data samples from the query set, because we need to use their labels to evaluate the loss and train the model. To overcome this issue, for the data sample j in the query set, we replace $o(y_j)$ in Eq. (5) by a uniform distribution over all activity classes to generate a type-two vertex as follows:

$$\text{Type-two vertex:} \qquad v_j =< r(x_j),\ \frac{1}{k} \times I_k >, \tag{6}$$

where k is the number of classes that the GNN model recognizes and I_k is an identity matrix of size k. The rationale is that we pretend not to know the activity class of each data sample in the query set. Therefore, their likelihood to each class is regarded to be the same initially. Later on, through training, their vertex vectors will be adjusted to become similar to the vertices from the support set for the same activity.

3.1.2 Graph Operations

After the vertices are constructed, we need to calculate the similarity between vertices and record it in the weights of the edges between the vertices. So we introduce an edge operation to compute edge weights.

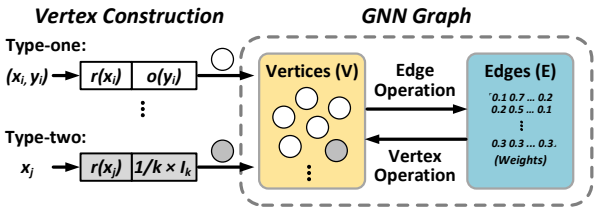


Fig. 3: Vertices and edges of the GNN graph.

1) Edge operation. Given two vertices v_i and v_j, we first compute the element-wise difference $\Delta v_{ij} = |v_i - v_j|$, where Δv_{ij} is also a one-dimensional vector with the same length as v_i or v_j, and each of its element records the difference between the corresponding elements between v_i and v_j. We then leverage a neural network to add non-linearity on Δv_{ij} to explore a distance metric. Specifically, we stack all Δv_{ij} together (from all vertex pairs) to form a virtual image $\{\Delta v_{ij}\}$. To extract local structural information from vertices, we then employ a convolution block $f_c(\cdot)$ to transfer $\{\Delta v_{ij}\}$ into a latent space and adopt an activation function $\sigma(\cdot)$ (*e.g.*, Leaky ReLU) to generate the distance metric. Therefore, given the set of all element-wise differences $\{\Delta v_{ij}\}$, the set of weights $\{w_{ij}\}$ for all pair of vertices v_i and v_j in the graph can be obtained at the same time by the following edge operation:

$$\{w_{ij}\} = \sigma(f_c(\{\Delta v_{ij}\})), \tag{7}$$

where Δv_{ij} is the difference between v_i and v_j when $i \neq j$ and $\Delta v_{ij} = 0$ when i equals to j to skip computing similarity to vertices themselves. Given a graph, applying the edge operation computes the weights w_{ij} for all edges e_{ij} in the adjacency matrix E in Figure 3. In general, the smaller w_{ij}, the higher the similarity between vertices v_i and v_j.

2) Vertex operation. After vertices are constructed, vertices of the same class may not have similar vertex values, especially for type-one and type-two vertices of the same class (since the latter one does not encode the actual label information). Hence, we need to update each vertex further so that eventually vertices of the same activity tend to be similar, while different activities end up with very different values.

In GNNs, vertex updates are usually performed by "averaging" the values of its neighbors [33], and we further consider the weight of each edge for weighted average, so that when the weight of the edge is relatively small, the values of the two vertices become closer. Therefore, we superimpose the current weights (similarities) on all vertices $E \times V$ and add them to all current vertices. Then, we design the following vertex operations to update vertices:

$$V' \quad = \quad \text{MLP}(E \times V) + V, \tag{8}$$

where $\text{MLP}(\cdot)$ is a multi-layer perceptron (MLP). This operation essentially propagates and updates feature information among all vertices with the help of edges.

3.1.3 GNN model

From the initial graph so far, we can obtain the updated graph by applying edge and vertex operations, as described

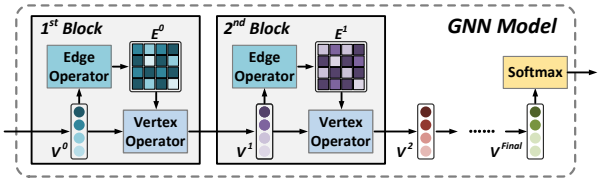


Fig. 4: GNN model composed of GNN blocks.

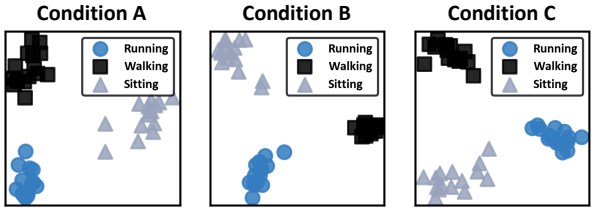


Fig. 5: Visualization for features of data samples of the same class in RoMF, which can be clearly distinguished under different sensing conditions.

above. We combine them together as one GNN block, and connect multiple GNN blocks to build a complete GNN model to increase the depth of the model for better performance. As our evaluation in §4 reveals, good recognition performance can be achieved as long as the number of GNN blocks is not particularly small, such as 1 or 3, and we employ five blocks in our current development.

1) Loss function. To perform recognition, we add a leaky ReLU [34] as activation function and a softmax layer at the end of the last GNN block, as shown in Figure 4. In this way, we can perform the backward propagation algorithm to update and train the model by calculating the loss value of the predicted data sample in the query set, that is, each data sample in the query set has a label, which we use to calculate the loss without encoding it to vertices in the graph (loss cannot be computed otherwise). The cross-entropy loss function is:

$$\mathcal{L}(\theta) \quad = \quad \hat{y}_i \times \log P(\hat{y}_i|x_i),$$

where $\hat{y}_i$ is the one-hot encoding of the predicted label y_i and θ represents the parameters of 1) ResNet $r(\cdot)$ in the vertex construction, 2) the neural networks in edge and vertex operations, and 3) the softmax layer. We can then update the GNN model by gradient descent $\theta \leftarrow \theta - \alpha \times \nabla\mathcal{L}(\theta)$, where α is the learning rate. When the training of a task converges, we can keep all the parameters in θ, but construct new vertices for the next task to continue training θ.

2) Model deployment. After the GNN model is trained (in §3.3), we can collect the same number of labeled data samples as the support set to form a reference set under the target condition, and organize each actual (to be recognized) sensing data as a test set. Then, the reference set samples and test set samples are used to construct type-one and type-two vertices, respectively. For each test set sample, our model can judge which reference data sample is most similar to it, and use this reference sample's label as the recognition result.

Our GNN model can work on new sensing conditions without fine-tuning for two reasons. First, meta-learning ensures that the data that the model needs to use at each stage of training (*i.e.*, support set and query set) and testing (*i.e.*, reference set and test set) come from the same conditions. Under the same sensing condition, data samples with the same label should be similar, and data samples with different labels should be different. Our method explicitly exploits this phenomenon and leverages this inherent invariance of data by directly comparing data similarities. Second, to make this comparison more reliable, we further encode the label information directly into the vertex construction as prior knowledge to effectively guide the model training, so that the model can effectively reveal and utilize this phenomenon under different sensing conditions, as shown in Figure 5.

3.2 Support for More Activity Classes

Before describing the details of how to train our GNN model using meta-learning, we first address a critical and practical issue related to the structural design of the GNN model. Assume that the number of activities to be recognized by the GNN model during training is k. If the number of activity classes under the new condition is K, *e.g.*, $K > k$, it means that more vertices need to be constructed for additional reference data samples (corresponding to these additional action classes), which requires changing the structure of the GNN model to accommodate these extra vertices. If the model structure changes, it must be updated before it can be used. To avoid modification of model in this case, we devise a decompose-and-vote block in RoMF.

3.2.1 Decompose-and-vote block

We introduce the design by taking the case where more classes need to be supported (*i.e.*, $K > k$), which can be extended to the case where $K < k$.

1) Design idea. Our main idea is to decompose the K-class recognition problem into a series of smaller k-class subproblems. Since each subproblem still targets k classes, which is the same number of classes as in training, we only need to use the trained k-class GNN model multiple times without modifying the structure of the model.[1] Therefore, we check all combinations (subproblems) of choosing k classes from K and infer the most likely class by examining the classification results of all $\binom{K}{k}$ subproblems via a voting mechanism.

Since data samples of the same class are similar to each other, our GNN model can reliably produce correct recognition results for the combinations, containing the class that the data sample to be recognized belongs to. On the contrary, if the actual class is not covered by a combination, the output will be an arbitrary class in the reference set, and the result will not focus on any class. Hence, we check all $\binom{K}{k}$ combinations for voting, and the correct class wins by accumulating votes for all combinations containing it.

2) Solution. We define a vector R to record the voting results, where R consists of K elements $\{R_l\}_{l=1}^{K}$, and each

1. Note that after the problem is decomposed in this way, although our GNN model needs to process different k activities each time, it can work normally because the meta-learning training in §3.3 takes this factor into account, that is, the classes involved in each task may themselves be different. In addition, our vertex design (directly encoding label information) further strengthens the model's ability to distinguish different classes.

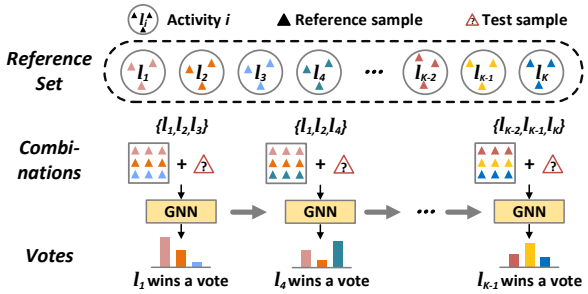


Fig. 6: Decompose-and-vote block when $k = 3$.

element R_l is initialized to zero. For each combination, we select all data samples in the reference set whose classes are covered by the k classes of that combination. We then perform classification by constructing type-one vertices for these samples and type-two vertex for the data sample in the test set. For the output class, we increment the corresponding element in R by one (*i.e.*, one vote). When we repeat this process for all $\binom{K}{k}$ combinations, as shown in Figure 6, the class $l^\star$ with the most votes is the recognition result of the sensing data sample in the test set:

$$l^\star = \arg\max_l \{R_l\}. \tag{9}$$

3) Early stopping. Since only combinations covering the class of the test set data sample can accumulate correct votes, we further propose an early stopping mechanism to reduce the execution overhead. Specifically, after running the voting mechanism for several combinations selected randomly, if we observe that some classes get much more votes than others, then they are more likely to be the recognition result, and we can focus on the combinations relate to these high-vote classes only to avoid many unnecessary computations. In principle, a class can receive no more than $\binom{K-1}{k-1}$ votes. Therefore, when any element in R gets sufficient votes, where we empirically choose half of that maximum value, we can pick the top-n classes with the most votes. Then we clear all the votes and recount from scratch by checking all the combinations that meet the following criteria: each combination encompasses all top-n classes ($k \geq n$), or each member within the combination belongs to the top-n classes ($k < n$). As shown in the evaluation (§4), in the case of a small n (*e.g.*, 3), the early stopping can reduce the computation amount by about 36% with almost no degradation in recognition performance.

4) Execution overhead. Leveraging the lightweight structure of GNN and our proposed early stopping mechanism, the overall execution overhead remains moderate, even as the number of combinations becomes large. Figure 7 shows the latency of RoMF to complete each recognition with different number of classes K, while maintaining the $k = 2$. It is evident that bigger K brings more combinations to examine, but the overall latency remains under 100 ms when $K = 22$. The early stopping mechanism can eliminates approximately 57% unnecessary computations.

3.2.2 Extension for fewer classes

In some cases, the number of classes under the new condition is reduced ($K < k$). This case can be supported during training by setting the value of k to be small, *e.g.*, $k = 3$.

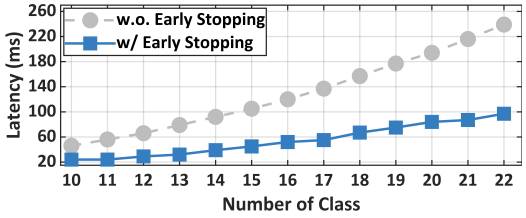


Fig. 7: System latency when the number of classes K changes (keeping $k = 2$).

Specifically, assuming that there are 10 classes of activities in the training dataset, we only need to train a 3-class GNN network, which is still able to recognize 10 classes through the decompose-and-vote block. In actual deployment, even if the number of activity classes under new conditions is reduced, for example, to 8 classes, we only need to modify K to 8. Therefore, our model can use the decompose-and-vote block as long as the number of classes is not less than k. Since k is small enough (*e.g.*, $k = 3$), it can meet most practical deployment requirements.

Final GNN configuration. While smaller k will result in more combinations, the GNN model itself will also be smaller, recognizing fewer classes each time, without increasing the overall inference time. According to the evaluation in §4, when $k = 3$, the inference time for an activity among eight classes can be finished within 80 ms on a desktop. Due to this capability, we set k small in RoMF, *e.g.*, $k = 3$ by default as studied in §4, which can support different numbers of activities flexibly after deployment without model fine-tuning.

3.3 Meta-Learning Framework

Unlike traditional supervised training, meta-learning generates a series of tasks to train the model. For each task, it utilizes a subset of the training dataset under certain conditions to form the support set and the query set. In WHAR, we take three important dimensions of environment, user, and activity as conditions for generating training tasks.

3.3.1 Task requirements

To train our GNN model, we introduce the following requirements to generate tasks for RoMF. We first randomly select a set of environment combinations, each of which is included in the training dataset. We denote all these environment combinations as $\mathcal{E}_t$, where $\mathcal{E}_t \subseteq \mathcal{E}$ and $\mathcal{E}$ denotes all possible combinations of environments in the training dataset. Similarly, we further select a set of activity combinations $\mathcal{K}_t$, where $\mathcal{K}_t \subseteq \mathcal{K}$ and $\mathcal{K}$ contains all possible combinations of activities involved in the training dataset. So the first requirement to generate each task t is:

$$\text{Criterion 1:} \quad \mathcal{E}_t \subseteq \mathcal{E} \text{ and } \mathcal{K}_t \subseteq \mathcal{K}.$$

Then, for each activity satisfying the first criterion, we randomly select data samples from different users (to cover diversity due to different users), and split them into a support set S_t^S (where m^S data samples) and a query set S_t^Q (with m^Q data samples) following the second requirement:

$$\text{Criterion 2:} \quad S_t^S \cap S_t^Q = \varnothing \text{ and } m^S + m^Q = |V|,$$

where there is no overlap between S_t^S and S_t^Q to ensure that the query set can effectively rehearse the learning performance, and the number of samples in both sets is equal to the number of vertices ($|V|$) of the GNN graph $G = (V, E)$. Based on these requirements, the support set and query set of each task face the same environments and activities but different users. This means that even within a single task, the model encounter different conditions, unlike traditional meta-learning methods. In real-world applications, users who contribute the reference set may differ from the actual users. Generating tasks in this way allows us to simulate this scenario.

3.3.2 Task generation

Following the above two criteria, we can generate various tasks to train the model by varying $\mathcal{E}_t$, $\mathcal{K}_t$, and the data samples in S_t^S and S_t^Q. Note that even if $\mathcal{E}_t$ and $\mathcal{K}_t$ are the same for two tasks, these tasks are still good for helping the model distinguish user diversity because the data samples are randomly selected from different users.

Finally, for each task, the number of samples in the support set (m^S) and query set (m^Q) is determined as follows:

1) Support set size m^S. Since our model recognizes k (*e.g.*, 3) classes each time, m^S is $m^S = k \times a$, where a is the number of data samples selected for each class. For training itself, there are no requirements on a. However, if we consider how the model is used after training, we can set a to be the same as the number of few-shot reference samples collected from the new condition. We set $a = 3$ by default in RoMF.

2) Test set size m^Q. The training itself has no special requirements on m^Q, and we mainly consider its setting in practice. After the GNN model is trained and deployed, the data samples in the test set are similar to the data samples in the query set during training. For each to-be-recognized data sample, the model should output a recognition result. Thus, we set $m^Q = 1$ for each recognition in the current design.

3.3.3 Summary

Finally, we walk through an example of how to set up our GNN model after training (using default system parameters). Assume that the number of activity classes in the training dataset is 10. We only train one GNN model for 3-class recognition. If the number of activities for a new user to be recognized in the new target environment is 12, we collect 3 data samples for each of the 12 activities to build a reference set. Later, when RoMF starts to recognize activities, the sensing signal is converted the sensing signature in the test set. Then RoMF runs the GNN model multiple times, each time for one of $\binom{12}{3}$ combinations, following the decompose-and-vote block. For each combination, 9 type-one vertices and 1 type-two vertex are generated to run the model. RoMF uses the result that has accumulated the most votes. After the current input is recognized, it is removed from the test set and the model waits for the next input.

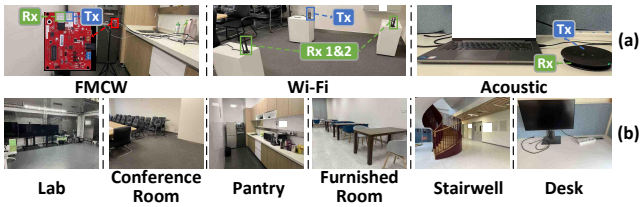


Fig. 8: (a) Our testbed built on commercial FMCW, Wi-Fi and acoustic devices. (b) Illustration of some of the sensing environments involved in our experiments.

4 Evaluation

4.1 Implementation

4.1.1 Testbeds and datasets

To thoroughly evaluate RoMF, we experiment with three popular wireless sensing signals, FMCW, Wi-Fi and acoustic waves. Further considering the generality of the design, we conduct experiments on both our testbeds and public datasets. Specifically, we built two testbeds for FMCW and acoustic sensing using commercial devices, as shown in Figure 8(a). For Wi-Fi, we mainly use the popular dataset Widar3.0 [35] for evaluation, and we also test RoMF (trained on Widar3.0) on our own Wi-Fi testbed. For all datasets, we ensure that the number of samples in each activity class is the same. This approach prevents overfitting due to data bias during model training and help the evaluation metric we use, accuracy, fairly represent the classification results across all categories. We detail below the setup and data collection process for all testbeds, which were ethically approved by the institute.

FMCW. We use a TI IWR1443BOOST radar to collect FMCW sensing data. The radar transmits 20 frames per second, each frame containing 128 chirps from the Tx antenna. In our experiments, for each chirp, the start frequency, bandwidth, and chirp duration are set to 77 GHz, 3.3 GHz, and 28 μs, respectively. We recruit 40 users to perform 12 activities in 11 different environments. These activities include: 1) jumping forward, 2) walking, 3) running, 4) squatting, 5) throwing, 6) walking in place, 7) running in place, 8) shaking hands, 9) chatting, 10) raising one hand, 11) using phone, and 12) push and pull. The environments are typical office and everyday life environments involving furniture and occlusions. Figure 8(b) shows some examples. For each activity, each user performed five times in each environment. After data collection, we randomly select the data of 20 users in 2 environments as the training dataset, and the data of the rest (new) 20 users are used for evaluation.

In addition, we further ask these 20 new users to a new environment (different from the above 11 environments) and perform 10 new activities, including 13) clapping, 14) raising both hands, 15) bowing, 16) lunging, 17) lying, 18) waving, 19) lifting one leg, 20) chatting, 21) sitting, and 22) drinking. Each activity is also performed five times. This set of data is used for evaluations involving new activities. Data collection for the following two signals follows a similar process.

Wi-Fi. For Wi-Fi, we mainly evaluate RoMF using the popular public dataset Widar3.0 [35]. It collects Wi-Fi Channel State Information (CSI) from Intel 5300 cards running at

5.825 GHz with a bandwidth of 20 MHz at a transmission rate of 1000 packets per sec. Widar3.0 contains sensing data of 22 everyday gestures/activities performed by 17 users in three environments. We randomly select 12 activity data of 4 users in 2 environments as the training dataset, and the rest are used for evaluation. For each activity, we also collect five samples for each user. In addition, we further collect data on the remaining 10 (=22 − 12) new activities of these 13 (=17 − 4) users (whose data is not present in the training dataset) in the last environment for evaluation involving new activities.

Acoustic. We use a pair of speaker-microphone to send and receive acoustic sensing data. Similar to existing acoustic sensing designs [30], the sender transmits continuous-wave sound at a constant frequency (17 kHz). For reliable sensing, the sender transmits at 16 different frequencies with an interval of 350 Hz. We repeat the first 12 gestures or activities in Widar3.0 with acoustic sensing on 16 users in 11 environments. For each activity, each user performs five times in each environment. After the data collection, we randomly select the data of 8 users in 2 environments as the training dataset, and the data of the rest 8 (new) users are used for evaluation. In addition, we further ask these 8 new users to a new environment to perform the remaining 10 activities in Widar3.0. This set of data is used for evaluations involving new activities under acoustic sensing.

	Total	Training	Env	Env+Usr	Env+Usr+Act
FMCW	12/40/22	2/20/12	10/20/12	10/20/12	1/20/10
WiFi	3/17/22	2/4/12	1/4/12	1/6/12	1/7/10
Acoustic	12/16/22	2/8/12	10/8/12	10/8/12	1/8/10

TABLE 1: Dataset partitioning under different unseen conditions (format: environment/user/activity).

4.1.2 Unseen conditions for evaluation

For comprehensive evaluation, we consider the following settings of conditions:

1) New environments only (Env): Only the environments are new and not present in training, while users and activities are seen during training. For example, for FMCW, we chose 12 activity data of 20 users in 2 environments as training dataset. Thus, for the condition with new environments only, we evaluate with the data of the same 12 activities from these 20 users in the other 10 new environments.

2) New environments plus users (Env+Usr): Only the activities are present in the training dataset, while the environments and users are new and unseen during training. Taking FMCW again as an example, we will use 12 activities of other 20 new users in 10 new environments for evaluation.

3) All factors are new (Env+Usr+Act): Users, environments and activities are all new, and their data are not present in training, *e.g.*, for FMCW, we use the data from 10 additional activities from 20 new users in the new environment.

The details of each sensing condition setting are summarized in Table 1. For each unseen condition, we use three data samples for each activity in the reference set (three-shot adaptation, similar to existing work [10], [11]), and one data sample currently to be recognized to form the test set.

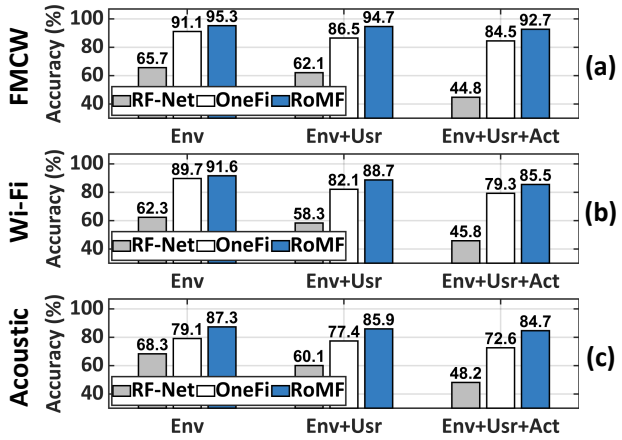


Fig. 9: Recognition accuracy comparison of the three methods.

It is worth to mention that Wi-Fi signals has the issue of unseen devices, which means different devices will have unique interference with the signals[36], [37]. To mitigate the impact of this issue, our setup binds the environment to specific devices, ensuring that the same environment does not encounter the problem of device replacement.

4.1.3 Model training

As mentioned in §2.2, we unify the input format of different sensing signals into a sensing signature in the form of a 2D matrix to facilitate development. In the current system, the matrix dimensions of FMCW, Wi-Fi, and acoustic are 128×512, 121×512, and 128×512, respectively. For Wi-Fi, if there are r receivers for recognition, *e.g.*, $r = 6$ in Widar3.0, we will concatenate r sensing signatures and feed the concatenated one into the model. With the decompose-and-vote mechanism, each GNN model performs k-class classification with an early-stopping parameter n. We study how the changes of these two parameters affect performance (§4.3), and then set $k = 3$ and $n = 3$ as default values. Since different sensing signals have different features, we use meta-learning to train three versions of RoMF (with the same model structure but different parameters), one for each sensing signal, developed using PyTorch [38]. Based on our task generation design, our GNN model contains 9 type-one vertices (3 classes $\times$ 3 samples per class) and 1 type-two vertex by default for all three sensing signals.

4.2 Overall Performance

In this evaluation, we compare three WHAR methods below:

- **RF-Net**: the state-of-the-art WHAR method that utilizes the metric-based meta-learning framework to fine-tune a pre-trained model for quick adaptation to unseen conditions [11].
- **OneFi**: the state-of-the-art Transformer-based WHAR method that involves dataset augmentation in the pre-training process to improve the performance of fine-tuned model for unseen gestures [13].
- **RoMF**: the method proposed in this paper.

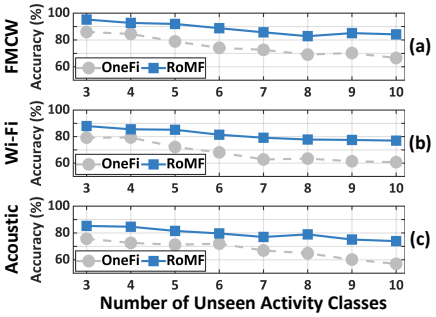


Fig. 10: Accuracy when the number of unseen activities changes.

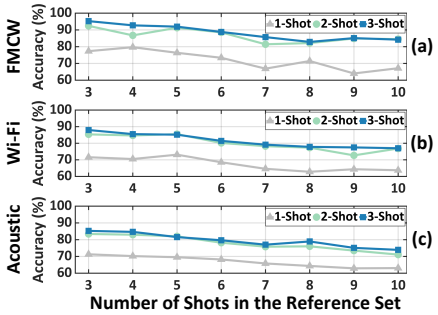


Fig. 11: Impact of number of shots of in the reference set.

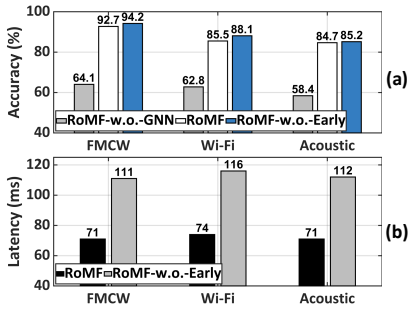


Fig. 12: Ablation study on (a) accuracy and (b) inference latency.

We train all three methods using the same training dataset presented in Table 1. For each unseen condition setting, we use data samples from the reference set to fine-tune RF-Net and OneFi before recognizing data samples from the test set. RoMF just takes the data samples in the reference set as input and performs recognition directly without any fine-tuning. When the number of activity classes changes under new conditions, both baselines require adjusting the network structure during fine-tuning, *i.e.* the MLP responsible for classification. In contrast, RoMF can use the Decompose-and-vote module to avoid any changes to the network structure.

There are some other recent WHAR designs [39], [40], [41] that follow the paradigm of fine-tuning a pre-trained model to handle new sensing conditions. However, they are mainly limited to certain specific radio frequency signals or applications. Therefore, we choose RF-Net and OneFi two more general methods in this evaluation.

Performance comparison. We first investigate the activity recognition accuracy of these three methods. Figure 9(a) shows the results using FMCW. When only the environment is new ("Env"), the accuracy of RF-Net is still limited, *e.g.*, 65.7%. Through our investigation, we found that its few-shot adaptation becomes less effective when the number of activity classes is large. OneFi improves the accuracy to 91.1%, while RoMF can go further to 95.3%. When further considering new users ("Env+User"), the accuracy of each method drops slightly, but RoMF can still maintain a high accuracy of 94.7%. When further including new activities ("Env+User+Act"), we introduce only four new classes here and change their number in the next experiment. OneFi maintains a good accuracy of 84.5% due to its effective learning, while RoMF can achieve 92.7% accuracy. Figures 9(b-c) further show the performance of the three methods using Wi-Fi and acoustic waves with similar conclusions.

Figures 9(b-c) further show the performance of the three methods using Wi-Fi and acoustic waves, which lead to similar conclusions as discussed with FMCW. Overall, RoMF improves activity recognition accuracy by 27.8–73.1% and 2.2–15.2% compared to RF-Net and OneFi, respectively.

Number of unseen activities. In Figure 10 we perform a detailed comparison when the number of unseen activities varies. The purpose of this experiment is to study how many unseen activity classes can be supported by RoMF without severe performance drop. Since RF-Net was not specifically designed for this setup, we only plot the results for OneFi and RoMF in Figure 10 for clarity. In this experiment, still under the setting of "Env+User+Act", we change the number of new activities from 3 to 10. When the number is small, the accuracy of the two methods is similar, such as 75.7–86.0% (OneFi) and 85.3–95.3% (RoMF) when the number is 3. As this number increases, recognition becomes more challenging, and the accuracy of both methods starts to drop. It can be seen that when the number is close to 10, the accuracy rate of OneFi drops to about 57.0–66.6%, but RoMF can maintain it to 73.9–84.2%. The performance is improved by 26.5–29.7%. Overall, RoMF can maintain a moderate performance even if the number of unseen classes becomes very large.

Ablation study. To gain further insight into the efficacy of the proposed technical design in RoMF, we conduct an ablation study by comparing the default version of RoMF with two other intermediate versions:

- **RoMF-w.o.-GNN**: This version still follows the overall design of RoMF, but replaces GNN with K-means to compare data similarity.
- **RoMF-w.o.-Early**: This version removes the early-stopping mechanism from the default version of RoMF.

Figure 12(a) shows that when our GNN model is replaced by the traditional clustering method, the accuracy drops significantly. This shows that our proposed GNN model can derive an effective distance metric that can reliably compare data similarity, which leads to a 32.2–44.8% improvement over "RoMF-w.o.-GNN". On the other hand, when the early-stopping mechanism is disabled, the accuracy of "RoMF-w.o.-Early" improves slightly since all combinations are checked before outputting the recognition result. From Figure 12(a), it can be seen that due to early stopping, the accuracy loss is very small, only 1.6–10.1%. However, Figure 12(b) shows that the inference latency due to early stopping can be significantly reduced, *e.g.*, by 36.2–36.6%. This ablation study shows that our technical design is effective in RoMF.

4.3 Micro-Benchmarks

Next, we further evaluate RoMF in different scenarios.

Number of GNN blocks. We first investigate how the number of GNN blocks in RoMF affects system performance. To this end, we change the number of GNN blocks from 1 to 7 for all unseen conditions. Through extensive evaluation in Figure 13, we can see that when the number of blocks is very

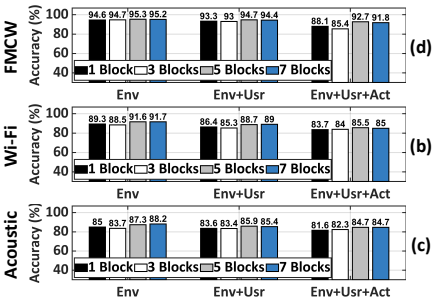


Fig. 13: Impact of GNN blocks.

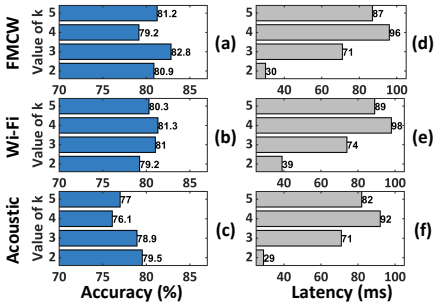


Fig. 14: Impact of parameter k.

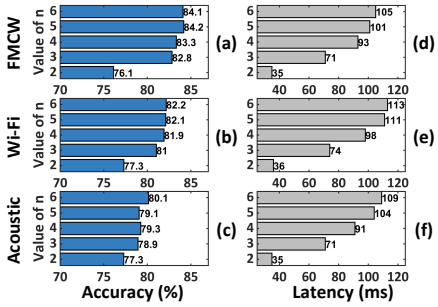


Fig. 15: Impact of parameter n.

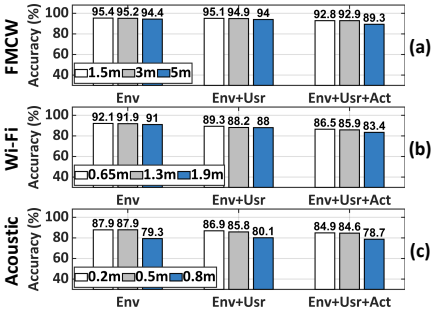


Fig. 16: Impact of different working distances.

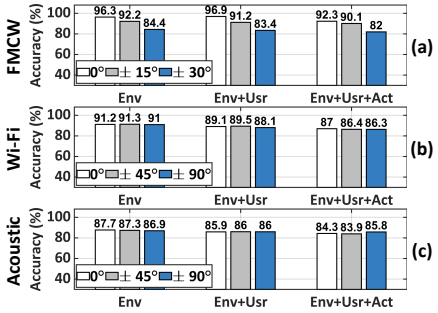


Fig. 17: Impact of different device orientations.

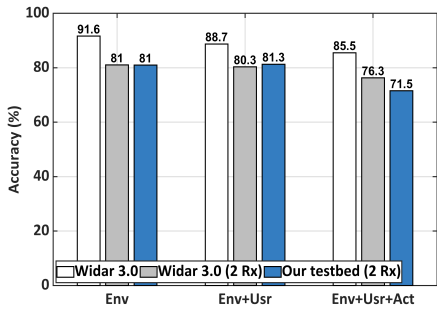


Fig. 18: Performance across different Wi-Fi datasets.

small, such as 1 or 3, the neural network has too few layers, making it insufficiently deep, which results in a relatively clear drop in accuracy for all cases. After the number of blocks is increased to 5, the performance is improved and becomes stable. As suggested by the result in Figure 13, we adopt five GNN blocks by default in the current RoMF.

Number of shots in the reference set. In this experiment, we investigate how the number of data samples (for each activity) in the reference set affects the system performance. To this end, we examine the performance of RoMF under the unseen condition setting of "Env+User+Act" by using one-shot, two-shots and three-shots of data samples in the reference set. Through extensive evaluation in Figure 11, we can see that when only one-shot is used, the accuracy is relatively low for all cases. However, after the number of shots is increased to two, the performance is improved significantly. By further increasing the number of shots, the performance improvement becomes small. Since three-shots the best performance in all cases in Figure 11, we configure the number of data samples in the reference set to be three for each activity.

Impact of decompose-and-vote parameters. With this mechanism, each GNN-model performs k-class recognition by checking the top-n classes for early stopping. We then study how k and n affect system performance.

When k is small, each GNN model recognizes fewer classes, which we find leads to more reliable recognition results and takes less time per execution. The downside is that more combinations need to be checked in the decompose-and-vote mechanism. In Figure 14(a-c), we show the accuracy of RoMF using different sensing signals when we change k from 2 to 5. Figure 14(d-f) further plot the corresponding inference time to complete each recognition at different values of k. We can see that smaller k tends to have better accuracy. On the other hand, smaller k produces less combinations, each model performs less inference time (as the model itself becomes simpler), so that Figure 14(d-f) show that the overall inference time is small.[2] Therefore, we prefer smaller k. Since $k = 3$ in general is more accurate than $k = 2$ across different signals, we default to $k = 3$ in RoMF.

In Figure 15, when n increases, both accuracy and inference time increase accordingly. However, we observe that after n is greater than 2, the improvement in accuracy is small, while the increase in inference time is still obvious. In our current implementation, we choose $n = 3$ as the default setting (because we need to run a lot of experiments). In actual deployment, people can choose a slightly larger n.

Impact of working distance. In this experiment, we investigate the effect of the working distance of the sensing device on the recognition performance. The default working distance between the center of the sensing area and the transmitter is about 3 m, 1.3 m, and 0.5 m for FMCW, Wi-Fi, and acoustic sensing, respectively. We further vary their working distances to study system performance. In Figure 16, for each set of three bars, the middle bar shows performance at the default working distance, and the left and right bars represent the results under reduced and increased working distances, respectively. For Wi-Fi, since the Widar3.0 dataset has detailed distance information for each data sample, we can select them according to the distance. In general, accuracy decreases as distance increases. Because the acoustic signal decays rapidly with distance, its accuracy drops more than the other two sensing signals. At longer working distances, FMCW and Wi-Fi can still achieve 89.3–95.4% and 83.4–91.0% accuracy in all cases, respectively.

2. In Figure 14(d-f), the total number of classes is eight. The latency of $k = 5$ is less than that of $k = 4$, as the additional latency (per model execution) due to increased model complexity is limited, but it has fewer combinations.

	Signature Generation	Inference	Overall
FMCW	110 ms	71 ms	181 ms
WiFi	520 ms	74 ms	594 ms
Acoustic	490 ms	71 ms	561 ms

TABLE 2: Latency to complete each activity recognition.

When the working distance is reduced, the accuracy of using each sensed signal can be further improved. Overall, the normal working distance of these sensing signals does not affect WHAR performance.

Impact of device orientation. In this experiment, we further investigate the effect of device orientation on recognition performance. For FMCW, direction is measured by the angle between the transmitting antenna and the user. For the other two signals, direction is measured by the angle of the user's position relative to the transmitter-receiver pair. It can be seen from Figure 17 that for Wi-Fi and acoustic sensing, the orientation has little effect on the recognition accuracy. However, when the orientation angle is large, the accuracy using FMCW drops significantly. This is due to the inherent principle of FMCW radar — when the target is not in front of the radar, the speed and distance changes caused by its motion become less sensitive and difficult to be captured by radar [21]. However, within a reasonable orientation angle, such as within 15°, the recognition performance is still good.

Different datasets. For Wi-Fi, we have so far mainly evaluated RoMF using the public dataset Widar3.0. In this experiment, we further evaluate the performance of RoMF (trained on Widar3.0) on our Wi-Fi testbed. We set up our Wi-Fi testbed according to Widar3.0, and the only difference is that our testbed has two receivers, while Widar3.0 involves six receivers. Therefore, we first test RoMF again on Widar3.0 (trained with data from six receivers on Widar3.0), but with data only from two receivers. Figure 18 shows that the accuracy drops to 76.3–81.0%. When RoMF is then directly tested on our test platform, a similar performance of 71.5–81.3% can be achieved. According to this result, we will further update our Wi-Fi testbed by increasing the spatial antenna gain to further improve its performance in the future.

System Overhead. Finally, we examine system overhead. Overall, RoMF is a lightweight WHAR design. The three versions of RoMF for FMCW, Wi-Fi and acoustic only contains 5.8, 6.2 and 5.8 million parameters, respectively. During our development, training of RoMF can be done in 8–10 hours. When RoMF works after training, it can complete each recognition within 181–561 ms, where the delay mainly comes from two parts: signal preprocessing to generate sensing signature and GNN-based inference. Specifically, it takes 110–490 ms to complete the sensing signature generation. Due to the small size of our GNN model, inference can be done within 71–74 ms, the power consumption and memory usage are below 120 watt and 1.1 GB, respectively. In contrast, the fine-tuning process requires 158 watts and 1.6 GB. This represents that 24% power consumption and 31% memory usage can be saved by RoMF by avoiding fine-tuning. Table 2 summarizes details using different sensing signals for eight classes on a desktop with an AMD 3700X CPU and NVIDIA 3090 GPU.

5 Points of Discussion

Scalability. In practical applications, the number of activities (K) to be classified frequently changes, which challenges the system's scalability. The design of RoMF can address this issue by training multiple versions of models, each with a different value of k. When the number of classes is large, we select the version with a larger k, and vice versa. This approach ensures performance while controlling system latency by managing the number of combinations checked by the Decompose-and-vote block.

Regression Tasks. With our current design, RoMF can only deal with classification tasks. In the field of wireless sensing, there are many regression tasks also face the issue of unseen conditions, such as localization and vital signal monitoring. GNN have the capability to handle regression tasks, as demonstrated in various applications like molecular property prediction, traffic flow prediction, and user influence prediction. Therefore, an important future work is to explore sensing methods for the unseen condition issue in regression tasks with GNN.

Deployment Potential. With the rapid advancement of edge AI hardware, the computational resource of current edge AI devices (*e.g.*, NVIDIA Jetson AGX Orin) are comparable to the hardware we used. Therefore, even though our experiments were not conducted on edge devices, the system overhead of RoMF still demonstrates its capability to be deployed on edge and perform recognition tasks efficiently.

6 Related Work

Wireless human activity recognition. Although traditional computer vision-based human activity recognition methods [42], [43], [44] have high accuracy, they are limited by several factors, such as lighting conditions, occlusion, and privacy concerns. Therefore, various wireless signals, including acoustic signals [30], [45], [46], millimeter waves [47], [48], [49] and Wi-Fi [50], [51], [52], are further explored to realize wireless human activity recognition (WHAR). However, the WHAR model generally cannot be used in new conditions (including environments, users, and activity classes), limiting practical deployment. To address this issue, Jiang *et al.*, Wang *et al.*, and Shi *et al.* [14], [53], [54] try to extract context/subject-independent features of data sharing under different conditions. Cheng *et al.* [35] enable cross-domain gesture recognition by defining gesture-specific features and body velocity curves. Meanwhile, with the advancement of meta-learning, MetaSense [10] uses it to efficiently adapt the recognition model to new users. RF-Net [11] combines a dual-path feature extractor and a metric-based meta-learning approach for few-shot activity recognition in new environments. OneFi [13] further proposes a one-shot learning algorithm to reduce the overhead of data collection under new conditions. However, these existing methods require fine-tuning the model, limiting the wide spread of WHAR. In RoMF, our GNN-based model can adapt to new conditions without fine-tuning the model.

Meta-learning. The purpose of meta-learning is to train a model with very few data samples [55], [56], [57]. It can be roughly divided into three types: model-based,

optimization-based, and metric-based. The model-based methods [57], [58], [59] learn a model that can adapt to new tasks by adjusting its parameters based on the similarity between the new tasks and the tasks used for training. Optimization-based methods [60], [61] aim to learn optimal training algorithms with well-initialized weights or learning rates. Metric-based methods [62], [63], [56] classify target data samples by comparing their similarity to labeled data samples with an appropriate distance metric. The design of RoMF in this paper belongs to a metric-based method under the meta-learning framework.

Graph neural networks. Graph Neural Network (GNN) [64], [65] consists of vertices and edges, and further utilizes edge operations and vertex operations to propagate messages and update vertex representations. GNNs are commonly used for few-shot learning tasks in computer vision [12], [66], where the goal is to learn new tasks with only a few labeled data samples. To the best of our knowledge, we are the first to combine GNNs and meta-learning in a WHAR system to adapt to new conditions without fine-tuning the model.

7 Conclusion

This paper introduces RoMF, a novel wireless-based human activity recognition system that can efficiently adapt to unseen conditions guided by few-shots of data samples, but avoids model fine-tuning. We observe and employ a key insight that is invariant across different sensing conditions, based on which we propose RoMF from a new learning perspective to minimize the effect of sensing conditions. Moreover, we design a decompose-and-vote mechanism to ensure that even if the number of activities changes under new conditions, fine-tuning can still be avoided. Extensive evaluations show encouraging performance gains of RoMF.

Acknowledgements

This work is supported by the General Research Fund (GRF) grants from Research Grants Council of Hong Kong (CityU 11213622) and the Shenzhen Science and Technology Program (Grant No.ZDSYS20210623092007023).

References

[1] Z. Chi, Y. Yao, T. Xie, X. Liu, Z. Huang, W. Wang, and T. Zhu, "Ear: Exploiting uncontrollable ambient rf signals in heterogeneous networks for gesture recognition," in *Proceedings of ACM SenSys*, 2018.

[2] A. Wang and S. Gollakota, "Millisonic: Pushing the limits of acoustic motion tracking," in *Proceedings of CHI*, 2019.

[3] H.-B. Zhang, Y.-X. Zhang, B. Zhong, Q. Lei, L. Yang, J.-X. Du, and D.-S. Chen, "A comprehensive survey of vision-based human action recognition methods," *Sensors*, 2019.

[4] J.-F. Hu, W.-S. Zheng, J. Lai, and J. Zhang, "Jointly learning heterogeneous features for rgb-d activity recognition," in *Proceedings of IEEE CVPR*, 2015.

[5] Q. Pu, S. Gupta, S. Gollakota, and S. Patel, "Whole-home gesture recognition using wireless signals," in *Proceedings of ACM MobiCom*, 2013.

[6] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: A survey," *Computer networks*, 2010.

[7] B. Jokanovic, M. Amin, and B. Erol, "Multiple joint-variable domains recognition of human motion," in *Proceedings of IEEE Radar Conference*, 2017.

[8] Z. Chi, Y. Yao, T. Xie, X. Liu, Z. Huang, W. Wang, and T. Zhu, "Ear: Exploiting uncontrollable ambient rf signals in heterogeneous networks for gesture recognition," in *Proceedings of ACM SenSys*, 2018.

[9] T. W. Hnat, V. Srinivasan, J. Lu, T. I. Sookoor, R. Dawson, J. Stankovic, and K. Whitehouse, "The hitchhiker's guide to successful residential sensing deployments," in *Proceedings of ACM SenSys*, 2011.

[10] T. Gong, Y. Kim, J. Shin, and S.-J. Lee, "Metasense: Few-shot adaptation to untrained conditions in deep mobile sensing," in *Proceedings of ACM SenSys*, 2019.

[11] S. Ding, Z. Chen, T. Zheng, and J. Luo, "Rf-net: A unified meta-learning framework for rf-enabled one-shot human activity recognition," in *Proceedings of ACM SenSys*, 2020.

[12] V. G. Satorras and J. B. Estrach, "Few-shot learning with graph neural networks," in *Proceedings of ICLR*, 2018.

[13] R. Xiao, J. Liu, J. Han, and K. Ren, "Onefi: One-shot recognition for unseen gesture via cots wifi," in *Proceedings of ACM SenSys*, 2021.

[14] W. Jiang, C. Miao, F. Ma, S. Yao, Y. Wang, Y. Yuan, H. Xue, C. Song, X. Ma, D. Koutsonikolas *et al.*, "Towards environment independent device free human activity recognition," in *Proceedings of ACM MobiCom*, 2018.

[15] A. Virmani and M. Shahzad, "Position and orientation agnostic gesture recognition using wifi," in *Proceedings of ACM MobiSys*, 2017.

[16] X. Wang, K. Sun, T. Zhao, W. Wang, and Q. Gu, "Dynamic speed warping: Similarity-based one-shot learning for device-free gesture signals," in *Proceedings of IEEE INFOCOM*, 2020.

[17] M. A. Jamal and G.-J. Qi, "Task agnostic meta-learning for few-shot learning," in *Proceedings of IEEE/CVF CVPR*, 2019.

[18] Y. Zhang, Y. Zheng, K. Qian, G. Zhang, Y. Liu, C. Wu, and Z. Yang, "Widar3. 0: Zero-effort cross-domain gesture recognition with wi-fi," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[19] C. Iovescu and S. Rao, "The fundamentals of millimeter wave sensors," *Texas Instruments*.

[20] H. Xue, Q. Cao, Y. Ju, H. Hu, H. Wang, A. Zhang, and L. Su, "M4esh: mmwave-based 3d human mesh construction for multiple subjects," in *Proceedings of ACM SenSys*, 2022.

[21] X. Zhang, Z. Li, and J. Zhang, "Synthesized millimeter-waves for human motion sensing," in *Proceedings of ACM SenSys*, 2022.

[22] D. Tahmoush, "Review of micro-doppler signatures," *IET Radar, Sonar & Navigation*, 2015.

[23] Y. Ma, G. Zhou, and S. Wang, "Wifi sensing with channel state information: A survey," *ACM Computing Surveys*, 2019.

[24] G. Chi, Z. Yang, J. Xu, C. Wu, J. Zhang, J. Liang, and Y. Liu, "Wi-drone: wi-fi-based 6-dof tracking for indoor drone flight control," in *Proceedings of ACM MobiSys*, 2022.

[25] Z. Chen, T. Zheng, and J. Luo, "Octopus: a practical and versatile wideband mimo sensing platform," in *Proceedings of ACM MobiCom*, 2021.

[26] J. Zhang, F. Wu, B. Wei, Q. Zhang, H. Huang, S. W. Shah, and J. Cheng, "Data augmentation and dense-lstm for human activity recognition using wifi signal," *IEEE Internet of Things Journal*, 2020.

[27] G. Laput, K. Ahuja, M. Goel, and C. Harrison, "Ubicoustics: Plug-and-play acoustic activity recognition," in *Proceedings of ACM UIST*, 2018.

[28] J. M. Sim, Y. Lee, and O. Kwon, "Acoustic sensor based recognition of human activity in everyday life for smart home services," *International Journal of Distributed Sensor Networks*, 2015.

[29] J. Lian, X. Yuan, M. Li, and N.-F. Tzeng, "Fall detection via inaudible acoustic sensing," *Proceedings of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2021.

[30] W. Wang, A. X. Liu, and K. Sun, "Device-free gesture tracking using acoustic signals," in *Proceedings of ACM MobiCom*, 2016.

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, 2008.

[32] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, 1958.

[33] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of ICML*, 2017.

[34] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of ICML*, 2013.

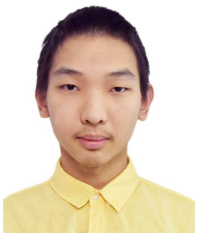
[35] Y. Zheng, Y. Zhang, K. Qian, G. Zhang, Y. Liu, C. Wu, and Z. Yang, "Zero-effort cross-domain gesture recognition with wi-fi," in *Proceedings of ACM MobiSys*, 2019.
[36] M. Kotaru, K. Joshi, D. Bharadia, and S. Katti, "Spotfi: Decimeter level localization using wifi," in *Proceedings of ACM SIGCOMM*, 2015.
[37] Z. Liu, G. Singh, C. Xu, and D. Vasisht, "Fire: enabling reciprocity for fdd mimo systems," in *Proceedings of ACM MobiCom*, 2021.
[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, 2019.
[39] J. Zhang, Z. Chen, C. Luo, B. Wei, S. S. Kanhere, and J. Li, "Metaganfi: Cross-domain unseen individual identification using wifi signals," *Proceedings of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2022.
[40] C. Feng, N. Wang, Y. Jiang, X. Zheng, K. Li, Z. Wang, and X. Chen, "Wi-learner: Towards one-shot learning for cross-domain wi-fi based gesture recognition," *Proceedings of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2022.
[41] Z. Ma, S. Zhang, J. Liu, X. Liu, W. Wang, J. Wang, and S. Guo, "Rf-siamese: approaching accurate rfid gesture recognition with one sample," *IEEE Transactions on Mobile Computing*, 2022.
[42] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *Proceedings of IEEE CVPR*, 2017.
[43] N. Hussein, E. Gavves, and A. W. Smeulders, "Timeception for complex action recognition," in *Proceedings of IEEE/CVF CVPR*, 2019.
[44] Z. Li, K. Gavrilyuk, E. Gavves, M. Jain, and C. G. Snoek, "Videolstm convolves, attends and flows for action recognition," *Elsevier Computer Vision and Image Understanding*, 2018.
[45] W. Mao, M. Wang, W. Sun, L. Qiu, S. Pradhan, and Y.-C. Chen, "Rnn-based room scale hand motion tracking," in *Proceedings of ACM MobiCom*, 2019.
[46] Y. Wang, J. Shen, and Y. Zheng, "Push the limit of acoustic gesture recognition," *IEEE Transactions on Mobile Computing*, 2020.
[47] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, "Soli: Ubiquitous gesture sensing with millimeter wave radar," *ACM Transactions on Graphics*.
[48] H. Liu, Y. Wang, A. Zhou, H. He, W. Wang, K. Wang, P. Pan, Y. Lu, L. Liu, and H. Ma, "Real-time arm gesture recognition in smart home scenarios via millimeter wave sensing," *Proceedings of ACM on interactive, mobile, wearable and ubiquitous technologies*, 2020.
[49] S. M. Kwon, S. Yang, J. Liu, X. Yang, W. Saleh, S. Patel, C. Mathews, and Y. Chen, "Hands-free human activity recognition using millimeter-wave sensors," in *Proceedings of IEEE DySPAN*, 2019.
[50] Y. Wang, J. Liu, Y. Chen, M. Gruteser, J. Yang, and H. Liu, "E-eyes: device-free location-oriented activity identification using fine-grained wifi signatures," in *Proceedings of ACM SenSys*, 2014.
[51] W. Jiang, H. Xue, C. Miao, S. Wang, S. Lin, C. Tian, S. Murali, H. Hu, Z. Sun, and L. Su, "Towards 3d human pose construction using wifi," in *Proceedings of ACM MobiCom*, 2020.
[52] D. Wu, D. Zhang, C. Xu, H. Wang, and X. Li, "Device-free wifi human sensing: From pattern-based to model-based approaches," *IEEE Communications Magazine*, 2017.
[53] Z. Wang, S. Chen, W. Yang, and Y. Xu, "Environment-independent wi-fi human activity recognition with adversarial network," in *Proceedings of IEEE ICASSP*, 2021.
[54] C. Shi, J. Liu, N. Borodinov, B. Leao, and Y. Chen, "Towards environment-independent behavior-based user authentication using wifi," in *Proceedings of IEEE MASS*, 2020.
[55] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, 2015.
[56] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, 2016.
[57] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of ICML*, 2017.
[58] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, 2017.
[59] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proceedings of ICML*, 2016.
[60] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," *arXiv preprint arXiv:1707.09835*, 2017.
[61] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," *Advances in neural information processing systems*, 2016.
[62] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of IEEE CVPR*, 2018.
[63] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, "Meta-learning for semi-supervised few-shot classification," *arXiv preprint arXiv:1803.00676*, 2018.
[64] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[65] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio *et al.*, "Graph attention networks," *stat*, 2017.
[66] J. Kim, T. Kim, S. Kim, and C. D. Yoo, "Edge-labeling graph neural network for few-shot learning," in *Proceedings of IEEE/CVF CVPR*, 2019.

Xiaotong Zhang received his B.E. degree in computer science and technology from Southern University of Science and Technology in 2019. He is currently a joint Ph.D. student in computer science at City University of Hong Kong and Southern University of Science and Technology. His research interests include wireless sensing and mobile AI systems.

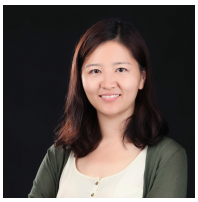
Qingqiao Hu received his B.E. degree in computer science and technology from Southern University of Science and Technology in 2021 and the M.S. degree in electrical and computer engineering from UCLA in 2023. He is currently a Ph.D. student in computer science at Stony Brook University. His research include deep learning, computer vision and technology making system automatic.

Zhen Xiao received the B.E. degree from City University of Hong Kong in 2018 and the Ph.D. degree from City University of Hong Kong, in 2023. He is currently a Postdoc in the Department of Computer Science, City University of Hong Kong. His research interests include wireless sensing, deep learning, and cyber security.

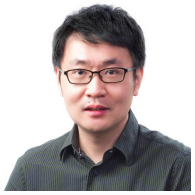
Tao Sun received his B.E. degree in Department of Computer Science and Engineering in 2022. He is currently a master student in Computer Science and Engineering, at Southern University of Science and Technology. His research interests include acoustic sensing and wearable devices.



Jiaxi Zhang received his B.E. degree in Department of Computer Science and Engineering in 2022. He is currently a joint Ph.D. in Computer Science and Engineering, The Hong Kong University of Science and Technology, and Department of Computer Science and Engineering, at Southern University of Science and Technology. His research interests include rehabilitation using wearable devices.



Jin Zhang is currently an associate professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen. She received her B.E. and M.E. degrees in electronic engineering from Tsinghua University, Beijing, in 2004 and 2006 respectively, and received her Ph.D. degree in computer science from Hong Kong University of Science and Technology, Hong Kong, in 2009. She was then employed in HKUST as a research assistant professor. Her research interests are mainly in mobile healthcare and wearable computing, wireless communication and networks, network economics, cognitive radio networks and dynamic spectrum management. She has published more than 70 papers in top-level journals and conferences. She is the Principle Investigator of several research projects funded by National Natural Science Foundation of China, Hong Kong Research Grants Council and Hong Kong Innovation and Technology Commission.



Zhenjiang Li received the B.E. degree from Xi'an Jiaotong University, China, in 2007, and the M.Phil. and Ph.D. degrees from the Hong Kong University of Science and Technology, Hong Kong, China, in 2009 and 2012, respectively. He is currently an Associate Professor with the Department of Computer Science, City University of Hong Kong. His research interests include Internet of Things, wearable and mobile computing, smart health, deep learning and distributed computing.