Graph-based Fingerprint Update Using Unlabelled WiFi Signals

KA HO CHIU, The Hong Kong University of Science and Technology, China

HANDI YIN, The Hong Kong University of Science and Technology (Guangzhou), China

WEIPENG ZHUO*, Guangdong Provincial/Zhuhai Key Laboratory of IRADS, and
Department of Computer Science, BNU-HKBU United International College, China

CHUL-HO LEE, Texas State University, United States

S.-H. GARY CHAN, The Hong Kong University of Science and Technology, China

WiFi received signal strength (RSS) environment evolves over time due to the movement of access points (APs), AP power adjustment, installation and removal of APs, etc. We study how to effectively update an existing database of fingerprints, defined as the RSS values of APs at designated locations, using a batch of newly collected unlabelled (possibly crowdsourced) WiFi signals. Prior art either estimates the locations of the new signals without updating the existing fingerprints or filters out the new APs without sufficiently embracing their features. To address that, we propose GUFU, a novel effective **g**raph-based approach to **u**pdate WiFi **f**ingerprints using **u**nlabelled signals with possibly new APs. Based on the observation that similar signal vectors likely imply physical proximity, GUFU employs a graph neural network (GNN) and a link prediction algorithm to retrain an incremental network given the new signals and APs. After the retraining, it then updates the signal vectors at the designated locations. Through extensive experiments in four large representative sites, GUFU is shown to achieve remarkably higher fingerprint adaptivity as compared with other state-of-the-art approaches, with error reduction of 21.4% and 29.8% in RSS values and location prediction, respectively.

CCS Concepts: • **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → **Machine learning**.

Additional Key Words and Phrases: WiFi Fingerprinting, Fingerprint update, Graph Neural Network, Crowdsourcing

1 INTRODUCTION

A WiFi fingerprint is defined as the received signal strength (RSS) values of the WiFi access points (APs) at a location. With the proliferation of APs and WiFi-enabled mobile devices [25, 32, 45, 50], WiFi fingerprints

---

*Corresponding author.

Authors' Contact Information: Ka Ho Chiu, khchiuac@connect.ust.hk, The Hong Kong University of Science and Technology, Hong Kong, Hong Kong, China; Handi Yin, hyin335@connect.hkust-gz.edu.cn, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong, China; Weipeng Zhuo, weipengzhuo@uic.edu.cn, Guangdong Provincial/Zhuhai Key Laboratory of IRADS, and Department of Computer Science, BNU-HKBU United International College, Zhuhai, Guangdong, China; Chul-Ho Lee, chulho.lee@txstate.edu, Texas State University, San Marcos, Texas, United States; S.-H. Gary Chan, gchan@ust.hk, The Hong Kong University of Science and Technology, Hong Kong, Hong Kong, China.

Table 1. Number of APs over a two-month period in a campus.

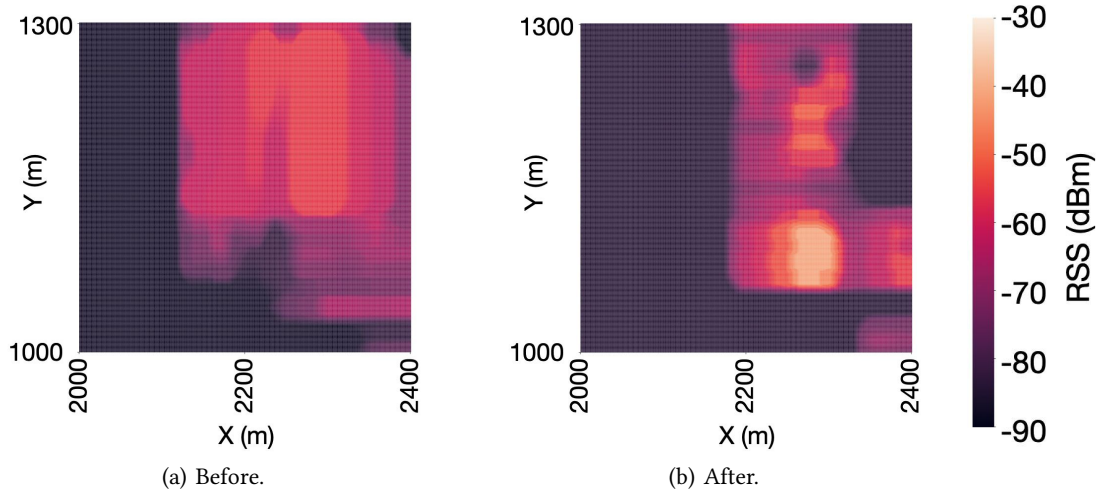| Before	After	Overlapped/Shared APs
494	517	193

(a) Before.

(b) After.

Fig. 1. Comparison of signal distributions from AP with SSID 34F8E7C0C583 over two months in a campus.

have found extensive applications in indoor navigation [7, 8, 12, 23, 30], WiFi service analysis [6, 27, 31, 36], geofencing [13, 16, 24, 26, 51], healthcare [11, 35, 49], etc.

A Wifi fingerprint database is often initially obtained with a full site survey where WiFi signals, given by AP IDs and their RSSs, are collected at designated locations (for example, over a grid size of a few meters). As WiFi environments evolve due to factors such as AP power adjustment, re-location, installation and removal, the existing fingerprint has to be updated over time to maintain service quality [10, 15, 20, 34, 37, 40]. Table 1 shows a WiFi environment on a campus that undergoes renovations, showcasing that the number of detected APs from multiple floors can significantly change over a two-month period. While some APs remain constant, numerous APs are removed, and a significant number of new APs are discovered. Figure 1 demonstrates an RSS heatmap of an AP that remains in the same area throughout this period, highlighting the notable differences in the spatial distribution of signals. These differences indicate that the AP power and/or location may have been changed.

In order to update an existing fingerprint database, traditionally new blanket site surveys are conducted regularly. As such surveys are labor-intensive and time-consuming, crowdsourcing approaches have been proposed, where new signals are collected spatially and randomly over time by mobile user devices in the venue *without* any location labels. These new unlabeled signal samples, sampled effortlessly at different locations, are processed periodically in batches, typically once every week or so, to update the existing fingerprints (RSS at existing designated locations).

In this work, we study the challenging problem of how to effectively update an existing fingerprint database given a batch of unlabelled (crowdsourced) signals that may consist of new APs. Our goal is to make full use of the signal features, including those of the new APs, to accurately update the signal vectors at the designated locations of the existing fingerprints.

Previous works on fingerprint update often estimate the locations of the newly collected signal samples and add them as new fingerprints into the existing database [1, 22, 33, 47]. While the new signals are included, the existing, possibly outdated, fingerprints, however, remain *unchanged* in the database. These outdated fingerprints can contaminate the database rather quickly over time. To tackle that, some recent approaches update the existing fingerprints by merging the new signals into the existing ones [3, 9, 20, 37, 39]. These methods, however, consider only the common APs without the *new* APs in the crowdsourced samples. This adversely affects the update process, because the information or features of the new APs have not been embraced or exploited.

We propose GUFU, a novel effective **g**raph-based approach to **u**pdate WiFi **f**ingerprints **u**sing unlabelled signals with possibly new APs. To the best of our knowledge, this is the first work on how to effectively update the existing (aged) fingerprints with a batch of newly collected unlabeled signals by considering all APs, including the shared and new ones.

GUFU consists of offline and online stages. In the offline stage, GUFU is initialized with a bootstrap site survey, where fingerprints are collected at designated grid locations, typically a few meters apart. Using the fingerprints, GUFU trains its two components, namely, a feature extractor and a graph neural network (GNN). The feature extractor extracts signal features from RSSs. The extracted features are then used to create a weighted graph formed by AP nodes and sample nodes, in which those features are fed as the initial sample node features. Subsequently, edges are created between the two types of nodes based on the signal strengths, while AP node features are initialized as the weighted average of the features from neighboring sample nodes. Additionally, observing that similar signal features likely indicate location proximity, extra virtual edges between sample nodes are added to the graph. The GNN is finally trained on this weighted graph, utilizing both node features and edge features, which are determined through weighted feature aggregation between the nodes.

After the offline stage, GUFU then enters into the online stage where the fingerprints evolve with the WiFi environment based on batches of unlabeled (crowdsourced) signals collected over time. New sample nodes for the graph are created using the shared APs between the existing fingerprints and each new batch of signals, along with nodes for any new APs that appear only in the new batch. The trained feature extractor and GNN are then applied to get those new nodes' features. These features are subsequently used in two MLPs to update the existing node features. The updated features are finally used to amend the corresponding signal strengths in the existing fingerprints.

To effectively address the evolving features of both new and outdated APs, GUFU employs a novel edge prediction algorithm to update the edges in the graph. This algorithm establishes new edges between the new APs and the labelled samples while removing potentially outdated edges. With these updated edges, the GNN can be updated to have its node features better represent the signal dynamics. Then by utilizing the trained feature extraction and node feature aggregation, the features of the new APs can also be leveraged to update the RSS values of the existing fingerprints.

Our contributions are summarized as follows:

- We propose GUFU, a novel graph-based approach to update WiFi fingerprints using unlabeled (crowdsourced) signals with possibly new APs. GUFU effectively amends each of the RSS values in the existing fingerprints at designated locations by embracing the features of the new APs.
- We propose a novel edge prediction module in GUFU to incorporate the features of the new APs. This module establishes connections between the new APs and existing samples in GNN to greatly improve the effectiveness of fingerprint updates. In addition to embracing new APs, this module can also remove existing yet outdated APs and their associated edges via a forgetting mechanism.
- We conduct extensive experimental studies on GUFU in four different major sites, namely, a campus building, a hospital and two shopping malls, over a long period of time (eight months). Our experimental results demonstrate that the fingerprints are effectively updated over time as the WiFi environment evolves.

GUFU outperforms state-of-the-art algorithms significantly, reducing 21.4% in RSS error and 29.8% in signal location error.

The rest of this paper is organized as follows. After reviewing the related work in Section 2, we overview GUFU in Section 3. In Section 4 we discuss the offline training of GUFU given an initial bootstrapped fingerprints by a site survey. In Section 5 we present the batch update of fingerprints for existing APs, while in Section 6 We present the fingerprint update with AP changes, i.e., addition and removal of APs. We discuss experimental results in Section 7 and conclude in Section 8.

2 RELATED WORK

Database update by appending new fingerprints: Many previous approaches predict locations for new signals and add them to the existing database. For the prediction task, earlier works [17, 38, 41] leverage additional information from IMU sensors or cameras in addition to WiFi signals. Other recent studies [1, 3, 4, 18, 28, 29, 33, 42, 43, 47] focus on WiFi signals and aim to train a WiFi-based classifier to find the most similar fingerprint records in the database for each new signal. For example, CNNLoc [33] combines a one-dimensional CNN with a stacked autoencoder for classifying WiFi signals. WiDeep [1] utilizes a similar autoencoder model and combines it with a noise injector to better extract AP-invariant features. FIDo [3] uses WiFi channel state information and builds a variational autoencoder to classify WiFi signals. WiDAGCN [47] models APs and signal samples into a graph and utilizes graph attention to model similarities between existing and new signals. These approaches add new signals with their predicted locations into the database. However, the existing and possibly outdated records in the database remain *unchanged*, which deteriorates the quality of fingerprints over time. By contrast, GUFU updates the existing (aged) fingerprints by estimating the RSS values that have been possibly outdated for each designated location in the database so that every record in the database can be up to date.

Database update by fusing new signals with existing fingerprints: Aged signals for the existing fingerprints and newly crowdsourced ones obtained from close locations may follow similar distributions. Hence, recent approaches [2, 3, 20, 21, 34, 37, 39, 46] attempt to fuse similar signals collected at *different* times to update fingerprints. For example, TransLoc [34] and iToLoc [20] aim to extract temporal-invariant features to train a classification model such that similar signals can be merged. In addition, Fidora [39] reconstructs old and new signals together via semi-supervised learning and updates signals in the old fingerprints. Another recent work MTDAN [37] further utilizes multi-target domain adaptation for signal updates. These approaches, however, only consider the shared APs that appear in the existing fingerprints. As a result, they have not fully exploited the signal features of the new APs, which could take a significant share in the newly collected signals, as shown in Table 1. By contrast, we propose an edge prediction module in GUFU to associate the new APs with the existing fingerprints. This allows us to leverage the features from the new APs in the process of updating the aged fingerprints.

Graph-based radio frequency (RF) signal modeling: RF signals such as WiFi signals are traditionally processed in the form of fixed-length vectors or matrices. This approach, however, may suffer from the missing-value problem [53]. Specifically, not all the signals from the APs at a location may be fully scanned, thereby leaving several entries in the vectors or matrices empty. These missing entries are usually filled with arbitrarily small values, but they may introduce unintended artifacts in the feature learning process. To solve this problem, several recent studies [5, 44, 47, 48, 52, 53] utilize a graph to model the RF signals. Zhang et al. [48] propose to use a homogeneous graph to model the relationships among signal samples, while WiDAGCN [47] models APs and signal samples separately. Besides utilizing a bipartite graph model with APs (given by MAC addresses) and signal samples being two different types of nodes, GUFU creates virtual edges to better capture their similarity. With this novel design, GUFU is able to effectively propagate possible changes in RSS values via edges in the graph and in turn better update the fingerprints.
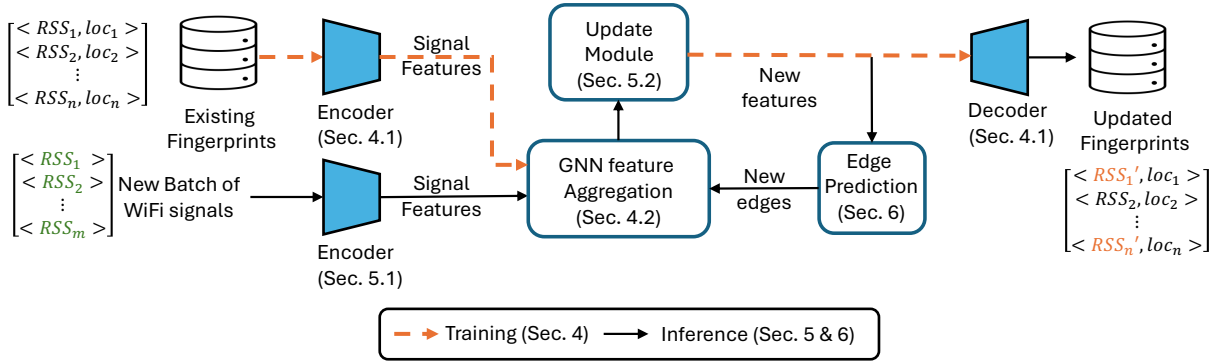
Fig. 2. A system overview of GUFU.

3 SYSTEM OVERVIEW

We overview GUFU in Figure 2. GUFU is bootstrapped with an existing fingerprint database, obtained through surveys at designated locations. This database is used for initialization, in which the RSS feature extractor is trained (Section 4.1) and the GNN is constructed using the signal features obtained by the trained feature extractor (Section 4.2). After that, given a newly collected batch of unlabeled signals, GUFU checks whether it has new APs apart from those in the existing fingerprints. For those existing APs, their signal features are obtained via inference through the GNN (Section 5.1), and new corresponding signal nodes are added to the GNN to infer the locations of new samples and the updated feature vectors for the signals (Section 5.2). For the new APs, an additional edge predictor is trained (Section 6.1) to create edges between those new APs and existing samples in the GNN. At the same time, a forgetting mechanism is applied to remove edges connected to potentially outdated APs as well as the nodes that correspond to the outdated APs (Section 6.2). These changes in AP nodes and edges enable the feature extractor and the GNN to be updated, and the updated feature vectors can be obtained from the updated GNN. Finally, those updated vectors are fed to the decoder for outputting the updated signal strengths in the existing fingerprint. To help better understand the operations of GUFU, in Table 2, we collect all the key notations used throughout the paper.

4 TRAINING WITH BOOTSTRAPPED FINGERPRINTS

In this section, we introduce the initialization of GUFU during the offline stage. GUFU is bootstrapped using a set of fingerprints collected through a site survey. These fingerprints serve as the input for training the RSS feature extractor (Section 4.1). After training, the extractor can generate the embeddings for any fingerprints. The embeddings of the existing fingerprints are then used to train a GNN on a weighted graph, along with additional virtual edges (Section 4.2).

4.1 Encoder and Decoder for Feature Extraction

The existing WiFi fingerprint database consists of signals and their corresponding location labels, typically obtained through surveys at predetermined locations. Each signal sample is obtained from a WiFi scanning and contains pairs of detected MAC addresses (MACs) along with their associated received signal strength (RSS) values. Let $Y \in \mathbb{R}^{N \times 2}$ represent the set of locations in the fingerprint database, where N is the number of samples with location labels. Additionally, let $X \in \mathbb{R}^{N \times n_s}$ denote the corresponding set of RSS values, where n_s is the total number of MACs detected at each location. If a particular MAC is not detected in a sample, we assign its corresponding entry in X a value of -120 dBm, which is the common practice in recent studies [47, 48, 52, 53].

Table 2. Summary of notations.

X	Set of RSS values in the fingerprint database
Y	Set of locations in the fingerprint database
U	Set of RSS values in a new batch of signals
Z_X	Set of signal features for X
Z_U	Set of signal features for U
V	Set of nodes in graph $\mathcal{G}$
v_x	A sample node in V, with $x \in X$
v_m	An AP node in V
z_x	Node feature for node v_x
z_m	Node feature for node v_m
E	Set of edges in graph $\mathcal{G}$
$E_{virtual}$	Set of virtual edges created in graph $\mathcal{G}$
z_{xm}	Edge feature for edge $e_{xm} \in E \cup E_{virtual}$
W	Set of edge weights in graph $\mathcal{G}$
$\mathbb{W}_0, \mathbb{W}_1, \mathbb{W}_2$	Trainable weights in the GNN
$G(v)$	Goodness score for a node v in graph $\mathcal{G}$
$F(v)$	Fairness score for a node v in graph $\mathcal{G}$

To normalize the RSS values to $[0, 1]$, we add an offset of 120 dBm to each entry in X and divide the resulting value by 120. We refer to X as the set of *normalized* RSS values for the rest of this paper.

In addition to the initial fingerprint database, RSS values are continuously collected in a crowdsourced manner, without accompanying location labels. A new batch of unlabeled samples becomes available at regular intervals, e.g., once a week. Let $U \in \mathbb{R}^{K \times n_b}$ represent the set of RSS values for this new batch, where K is the number of new samples and n_b is the total number of MACs detected in this batch. As done for the initial fingerprint database, if a MAC is not detected in a new sample, its corresponding RSS value is filled with -120 dBm. To ensure consistency, we also apply the same normalization process to each entry of U.

To reduce the dimensions of RSS values and extract fixed-length feature vectors for representing the samples in the initial fingerprints as well as from new batches of samples, we employ an autoencoder as our feature extractor. The autoencoder comprises an encoder $\mathbb{E}(\cdot)$ and a decoder $\mathbb{G}(\cdot)$, and its architecture is illustrated in Figure 3. For the RSS-value set X, we generate its corresponding feature set Z_X by passing it through the encoder $\mathbb{E}(\cdot)$. This operation can be expressed as

$$Z_X = \mathbb{E}(X). \tag{1}$$

The extracted feature set Z_X then goes through the decoder $\mathbb{D}(\cdot)$ to generate the following recovered RSS set:

$$\hat{X} = \mathbb{D}(Z_X). \tag{2}$$

The autoencoder network is trained by minimizing the following loss of the feature extraction:

$$\mathcal{L}_F = \|X - \hat{X}\|_F,$$

where $\| \cdot \|_F$ is the Frobenius norm.

Once the autoencoder is trained, we obtain the output feature set Z_X, which represents the features of the signals in the fingerprint database. When a new batch of WiFi signals U is collected, we can extract their features
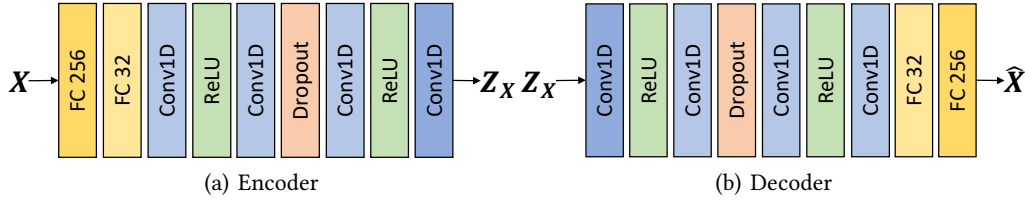
Fig. 3. Structure of the autoencoder.

by employing the same encoder. This operation can be expressed as

$$Z_U = \mathbb{E}(U). \tag{3}$$

4.2 Graph Formulation and GNN Training

Given the RSS-value set X and its corresponding extracted feature set Z_X, we construct a graph $\mathcal{G} = (V, E, W)$, where V represents the nodes, E denotes the edges, and W contains the corresponding edge weights. For each sample x in X, which is a row vector, we model it as a sample node v_x in the graph. Additionally, we model each detected MAC address in X as an AP node. If an AP node v_m is detected in the sample x, its corresponding node in the graph is the sample node v_x. Consequently, an edge $e_{xm} \in E$ is established between v_x and v_m. Its edge weight is defined as a non-negative function of the detected RSS value. Specifically, the edge weight w_{xm} is calculated as $x_m + c$, with $c > \max\{|X_{ij}|, \forall i, j\}$, where x_m represents the RSS value of AP m in x, and X_{ij} denotes the RSS value of AP j in WiFi signal sample i. We here set c to 120, which is a common practice for adding RSS values to graphs in [47, 48, 52, 53]. Note that if a MAC address is not detected in a WiFi sample x, its corresponding value is filled with -120 dBm. In such a case, no edge exists between the corresponding pair of nodes in the graph.

Each sample node v_x in the graph is initially associated with the feature vector of x obtained by the feature extractor in Section 4.1. Letting z_x be the feature vector of x, it is a row vector in Z_X. We augment this z_x by concatenating z_x with its location label y. For simplicity, we continue to denote the augmented vector as z_x throughout the rest of the paper. Then for any AP node v_m, its node feature is initialized as the weighted average of the node features in its immediate neighborhood $\mathcal{N}(v_m)$, which are all sample nodes. This initialization reflects the spatial relationships among the nodes and is given by

$$z_m = \sum_{v_x \in \mathcal{N}(v_m)} \frac{w_{mx}}{\sum_{v_y \in \mathcal{N}(v_m)} w_{my}} z_x.$$

This approach effectively aggregates information from all neighboring sample nodes of each AP node, incorporating that information into the AP nodes.

Thus far, edges have only been established between sample nodes and AP nodes. However, to update the existing fingerprints, which consist of signal samples, it is important to emphasize the sample nodes that contain features for these signals. To enable direct interactions between similar signal samples, we create virtual edges, denoted as $E_{virtual}$, connecting these sample nodes. The similarity between signal samples is determined by the common AP neighbors that they share, as illustrated in Figure 4. This similarity can be quantified using cosine similarity between each pair of sample nodes, which is calculated as the dot product of their respective node feature vectors.

To enable direct connections between similar sample nodes, one potential approach is to identify the k nearest neighbors of each node based on their cosine similarity values. However, it remains a challenge how to determine
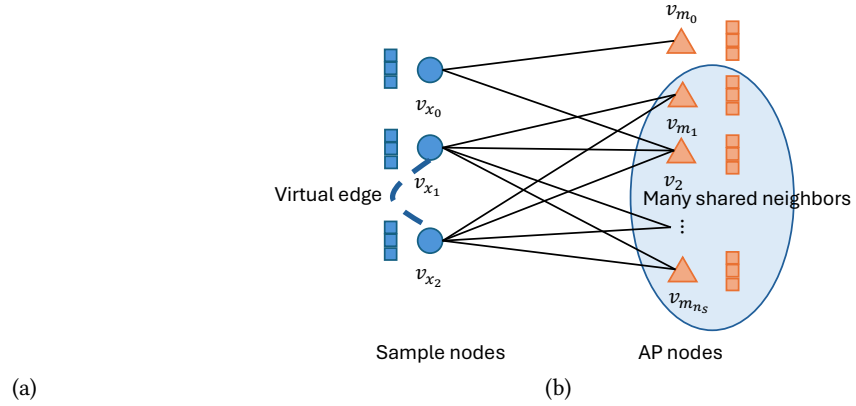
Fig. 4. Virtual edge creation. (a) Before virtual edge creation, edges only exist between sample nodes v_x's and AP nodes v_m's; (b) After creation, virtual edges are added among sample nodes having many shared AP neighbors like v_{x_1} and v_{x_2}.
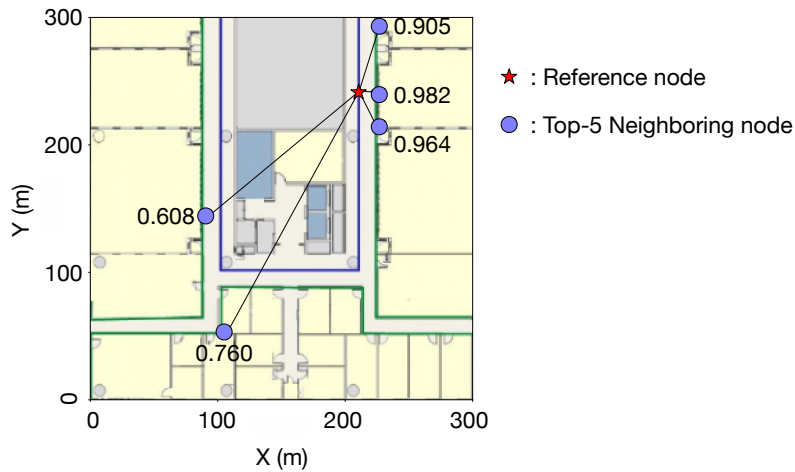

Fig. 5. Feature similarity between a reference node and its neighboring nodes can be low even for top-5 neighbors.

the appropriate value of k, as a neighboring node with low similarity could still appear in the top-k list regardless of the choice of k. This problem is illustrated in Figure 5, where the top five neighbors of a WiFi sample node are identified, but two of them display notably low similarity. To mitigate this problem, GUFU creates virtual edges using a thresholding method with a predefined threshold σ on the cosine similarity values. In this process, only pairs of sample nodes with similarity values greater than σ can be connected by virtual edges.

To learn a node embedding (or an updated node feature vector) for each node in the constructed graph $\mathcal{G}$, we construct a GNN by employing its training process based on GraphSAGE [14]. The principle behind GraphSAGE is that the embedding of each node is learned by aggregating information from its sampled one-hop neighborhood,

e.g., calculating the mean value of the node embeddings within the neighborhood. By employing L layers of aggregation, the embedding of each node incorporates information from its L-hop neighbors. In GUFU, we adapt and modify the aggregation process as follows.

We first introduce an edge feature vector as an additional input to the aggregation process. For each edge $e_{uv} \in E \cup E_{virtual}$, we initialize its edge feature vector z_{uv} as the mean value of the node features of its connected end nodes u and v, i.e., $z_{uv} = (z_u + z_v)/2$. The node-wise and edge-wise features are then fed into the aggregation process. The aggregation process is summarized in Algorithm 1, where L denotes the number of aggregation layers, and $v\_edge(\cdot)$ represents the creation of virtual edges. Before the L layers of aggregation, virtual edges are constructed. The process at each layer can be divided into the following three steps:

Algorithm 1: GUFU: Training

Input: Graph $G(V, E, W)$; initialized node embeddings $z_v, \forall v \in V$; number of aggregation layers L.

Output: Node embeddings $z_v, \forall v \in V$.

1 $E_{virtual} = v\_edge(E, V, \sigma)$.   /* Create virtual edges between sample nodes.   */

2 $z_{uv}^0 \leftarrow \frac{z_u + z_v}{2}, \forall e_{uv} \in E \cup E_{virtual}$.

3 **for** $l = 1, 2, \cdots, L :$ **do**

4   **for** $v \in V$ **do**

       /* Aggregate node features using the features from the previous layer.   */

5        $z_{hv}^l = AGG\left(\phi(\mathbb{W}_0^l \cdot \text{CONCAT}(z_{uv}^{l-1}, z_u^{l-1})), \forall e_{uv} \in E \cup E_{virtual}\right),$

       /* Update node features.   */

6        $z_v^l = \phi\left(\mathbb{W}_1^l \cdot \text{CONCAT}(z_v^{l-1}, z_{hv}^l)\right)$

7        $z_v^l = \frac{z_v^l}{||z_v^l||_2}$

    end

8   **for** $e_{uv} \in E$ **do**

       /* Update edge features.   */

9        $z_{uv}^l = \phi\left(\mathbb{W}_2^l \cdot \text{CONCAT}(z_u^l, z_{uv}^{l-1}, z_v^l)\right)$

10        $z_{uv}^l = \frac{z_{uv}^l}{||z_{uv}^l||_2}$

    end

    end

11 **return** $z_v, \forall v \in V$.

For every node v, we first aggregate the embeddings (or feature vectors) from its sampled neighbors, where the embedding of each neighbor is concatenated with the embedding of its connected edge (Line 5). Subsequently, the aggregated embedding from the neighbors is concatenated with the node's original embedding (Line 6). This concatenated embedding is then normalized (Line 7) to yield a new node embedding. At the end of each layer, the embedding of each edge is updated by concatenating the new embeddings of its end nodes with its existing edge embedding (Line 9 and Line 10). The learnable weights $\mathbb{W}_0^l$, $\mathbb{W}_1^l$, and $\mathbb{W}_2^l$ are trained by minimizing the difference between nodes and their immediate neighbors in terms of their embeddings. In other words, the training loss is given by

$$\mathcal{L}_{\mathcal{G}} = -\sum_{e_{uv} \in E \cup E_{virtual}} \log(\phi(z_u^T z_v)), \tag{4}$$

where $\phi(\cdot)$ is an activation function. After the aggregation, to reduce the space required for maintaining the graph $\mathcal{G}$, the virtual edges are removed. This completes the initialization of GUFU.

5 FINGERPRINT UPDATE FOR EXISTING APS

In this section, we present the batch update of GUFU during the online stage. This update process involves the existing APs that occur in the existing fingerprint database. These APs include the shared APs, which are common to both the new batch of signals and the existing fingerprint database, as well as those that are missing from the new batch. The trained GNN processes the new signals and generates their corresponding embeddings (Section 5.1). Using those embeddings, an MLP-based update module is utilized for location predictions and fingerprint updates (Section 5.2).

5.1 GNN Feature Aggregation

When a new batch of signals U arrives, new sample nodes are created based only on the RSS values from the existing APs and added into $\mathcal{G}$. The initial node features of these new sample nodes, obtained by the trained encoder in Section 4.1, are represented by Z_U. Each column vector in Z_U corresponds to the embedding of a new sample node, which is then concatenated with a randomly assigned two-dimensional location label. This concatenation is the same as what was done in Section 4.2.

Let V_U represent the set of new sample nodes, and let V_X denote the set of existing sample nodes. Following the process in Section 4.2, a new set of virtual edges is created between all sample nodes in $V_U \cup V_X$ using their current node features. Subsequently, GUFU retrieves the updated node features for all sample nodes and existing AP nodes by executing Algorithm 1. In other words, the embeddings of nodes in $V_U \cup V_X$ and all relevant edge features are updated via the aforementioned aggregation process. In particular, the embeddings of new sample nodes, i.e., Z_U, are used to update the existing fingerprint database.

5.2 Update Module

To iteratively predict the locations of new samples and obtain updated features for existing samples in the fingerprint database, we introduce an update module consisting of two sequential MLP networks. For a new batch of samples, the first MLP network takes their feature vectors Z_U as input and predicts their corresponding location labels $\hat{Y}_U$. In addition, the second MLP network is responsible for taking the location labels of samples in the existing fingerprint database and producing their updated feature vectors $\hat{Z}_X$. Here we utilize Z_X to recover their corresponding *updated* RSS values $\hat{X}$. This process is achieved using the decoder as in Equation (2). The updated RSS values are then used to replace the old ones in the fingerprint database. After this update is completed, the new sample nodes, along with their connected edges, are removed from the graph. Consequently, the number of nodes of the graph $\mathcal{G}$ remains unchanged. In other words, each batch of new samples is utilized to update the fingerprint database without expanding its size.

The first MLP network is trained using the feature vectors of existing samples Z_X and their corresponding location labels Y_X in the fingerprint database. Specifically, the network is trained to minimize the following loss function, which measures the discrepancy between the predicted location labels $\hat{Y}_X$ and the true labels Y_X:

$$\mathcal{L}_{\mathcal{P}} = ||Y_X - \hat{Y}_X||_F. \tag{5}$$

Similarly, the second MLP network is designed to minimize the following loss function, which measures the discrepancy between the updated feature vectors $\hat{Z}_U$ (the output of the MLP network) and their corresponding ground truth vectors Z_U:

$$\mathcal{L}_{\mathcal{U}} = ||Z_U - \hat{Z}_U||_F. \tag{6}$$

It is worth noting that the two MLP networks are similar to an autoencoder. The first MLP network serves as the encoder, which is responsible for the location prediction. The second MLP network acts as the decoder, which is for the fingerprint update.

In addition to $\mathcal{L}_\mathcal{P}$ and $\mathcal{L}_\mathcal{U}$, we define a consistency loss among neighboring nodes in the graph to train the MLP networks with higher accuracy. The rationale behind this consistency loss is that the newly predicted location or updated feature vector of each node should not differ too much from its neighborhood. It has two parts. The first one is for the location prediction. The predicted locations of new sample nodes should have similar location labels to those of old sample nodes in their virtual neighborhoods. Thus, we define the following loss function:

$$\mathcal{L}_{CP} = ||Y_X A - Y_U||_F, \tag{7}$$

where A is a matrix with elements A_{ij} being $A_{ij} = 1$ for all $e_{uv} \in E_{virtual}$ and $A_{ij} = 0$ otherwise. The second part is for the feature-vector update. Similar to the first part, the updated feature vectors of (old) sample nodes in the database should be similar to the features of new sample nodes in their virtual neighborhoods, leading to the following loss function:

$$\mathcal{L}_{CU} = ||Z_X A - Z_U||_F. \tag{8}$$

To summarize, considering all four loss functions, we have the final loss function to train the update module for each batch of new samples, which is given by

$$\mathcal{L} = \alpha(\mathcal{L}_\mathcal{P} + \mathcal{L}_\mathcal{U}) + (1 - \alpha)(\mathcal{L}_{CP} + \mathcal{L}_{CU}), \tag{9}$$

where α is the regularization parameter.

After applying the MLPs for signal and location updates, we remove the virtual edges created for this batch of new signals to minimize the storage requirements for the graph $\mathcal{G}$, similar to the approach described in Section 4.2. Additionally, for every batch update, we update the trainable weights of the GNN by minimizing the loss function in Equation (4). This loss function captures the differences between the embeddings of recently created nodes and their neighbors, including both direct neighbors and those connected through virtual edges. Furthermore, since X's feature vectors Z_X are updated for each new batch and the corresponding APs may changed, the original autoencoder should also be updated to ensure that Z_X can be extracted from X. To resolve this, we retrain it using the following training loss:

$$\mathcal{L}_F = ||X - \hat{X}||_F + \frac{1}{2}(||\mathbb{E}(X) - Z_X||_F + ||\mathbb{D}(Z_X) - \hat{X}||_F). \tag{10}$$

6 FINGERPRINT UPDATE WITH AP CHANGES

In this section, we explain how GUFU updates the edges in the graph $\mathcal{G}$ according to AP changes, including the addition of new APs and the removal of existing ones. For a new batch of signals, GUFU first assesses whether any new AP nodes need to be created. If so, it creates the new AP nodes and utilizes the node features from the graph whose AP nodes only consist of the existing APs to determine where new edges should be established between the new AP nodes and existing sample nodes (Section 6.1). GUFU also evaluates whether any existing edges should be removed from the graph, regardless of the presence of new APs (Section 6.2). Throughout this process, the node features remain unchanged.

6.1 Edge Prediction for New APs

After updating the node features for the existing and new sample nodes $V_U \cup V_X$ using only the existing APs, GUFU checks for any APs, identified by their MAC addresses, that are detected in U but not present in X. If such APs are identified, new corresponding AP nodes are created in the graph $\mathcal{G}$ for those MAC addresses. Then, similar to the initial training process in Section 4.2, the initial feature vector for each newly added AP node is calculated as the weighted average of the node feature vectors of its immediate neighbors.

To effectively utilize incoming new samples for updating the signals in the existing fingerprint database—especially the RSS values from new APs—we introduce an edge prediction algorithm. This algorithm identifies potential
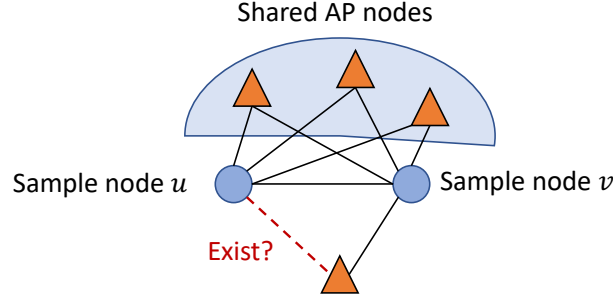
Fig. 6. Edge prediction.

missing connections between AP nodes and sample nodes, making it particularly useful for newly detected AP nodes and existing sample nodes.

As depicted in Figure 6, our edge prediction algorithm aims to predict missing edges between pairs of AP nodes and sample nodes that are not currently connected but are both present in the neighborhood of another sample node. As mentioned in Section 4.2, virtual edges are established between similar sample nodes. Hence, we search for node pairs consisting of an AP node and a sample node within the neighborhood of sample nodes that serve as endpoints of virtual edges. This edge prediction process allows us to discover potentially close AP nodes and sample nodes that are not yet connected in the graph.

In the algorithm, we adapt two metrics proposed in [19], namely, goodness and fairness, to assess edge weights in the graph and decide whether an AP node and a sample node should be connected. The goodness metric indicates how much a node is trusted by its neighbors as a similar node, while the fairness metric measures how reliable a node is in evaluating the goodness of its neighbors. It is important to note that goodness and fairness are interdependent. Specifically, if a node exhibits high goodness and high fairness, nearby nodes are more likely to connect with it.

To define the two metrics explicitly in our scenario, we define the goodness $G(v)$ of node v as the normalized weighted average of its neighbors' fairness values. The higher the value, the better the goodness. We also define the fairness $F(v)$ of node v as the normalized difference between the weights of its connected edges and the goodness values of its neighbors. The larger the difference is, the lower its fairness. We provide an illustrative example of goodness and fairness in Figure 7, where in our case the goodness and fairness are quite close to the edge weights. As illustrated in the figure, node u is considered good as the weighted average fairness of its neighboring nodes is close to 1. Node v is considered fair as the average difference in edge weights between it and its neighboring nodes is close to 0. As each node is associated with its d-dimensional embedding (feature vector), we define the goodness and fairness along each dimension, i.e., $G(v) = [g_1(v), \cdots, g_d(v)]$, and $F(v) = [f_1(v), \cdots, f_d(v)]$. Specifically, for each dimension i, we define $g_i(v)$ and $f_i(v)$ as

$$g_i(v) \quad = \quad \frac{1}{|N(v)|} \sum_{u \in N(v)} f_i(u)\overline{w}_{uv}, \tag{11}$$

$$f_i(v) \quad = \quad 1 - \frac{1}{2|N(v)|} \sum_{u \in N(v)} |\overline{w}_{uv} - g_i(u)|, \tag{12}$$

respectively, where $\overline{w}_{uv}$ is the normalized weight of edge e_{uv}, which is defined as $\overline{w}_{uv} = w_{uv}/w_{\max}$, with $w_{\max} = \max\{w_{ij}, \forall i, j\}$.

Our edge prediction algorithm is summarized in Algorithm 2. We first normalize the edge weights by the maximum edge weight in the graph (Line 1). The goodness and fairness vectors are both initialized as the updated
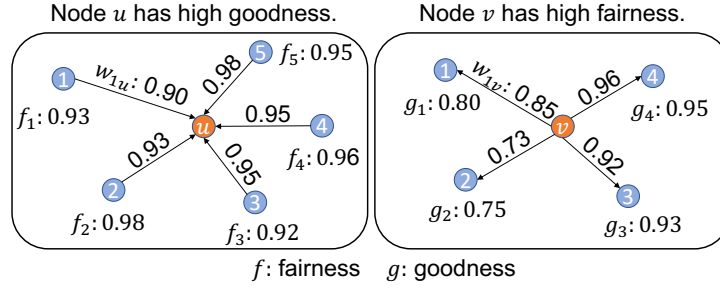
Fig. 7. An illustrative example of goodness g and fairness f.

node features (Line 2). We then iteratively update the goodness and fairness values for each node (Lines 5–9) until they converge. We next compute the predicted edge weights for potential edges, i.e., edges between AP nodes and sample nodes that are not connected but are both present in the neighborhood of another sample node (Line 12). If the predicted edge weight is greater than a (predetermined) threshold, then a new edge is added to the graph with the corresponding edge weight (Lines 13–15). The threshold is given by $\delta = 120 - \max\{|X_{ij}|, \forall i, j\}$. By setting this threshold, we can ensure that edges would only be created when the predicted weights are greater than $120 - c$, which equals 0 as we follow the common practice in [47, 48, 53] and set $c = 120$. Specifically, edges will only be created for those predicted to have non-negative edge weights, guaranteeing that all created edges possess valid, positive weights.

6.2 Forgetting Mechanism for Existing APs

In the process of updating fingerprints, it is important to consider not only the installation of new APs but also the removal of existing ones from a site. When an AP is removed, its signals will no longer be recorded, necessitating their removal from the fingerprint database. To tackle this issue, we introduce an edge removal process in our edge prediction algorithm, as outlined in Algorithm 2. During this process, GUFU identifies predicted edge weights that fall below the threshold for edge existence, which is the same threshold used for adding edges. If an edge's predicted weight is below this threshold, it is removed from the graph (Lines 16–18). Furthermore, if any AP node becomes disconnected from the rest of the graph as a result of the edge removal process, it indicates that the corresponding AP has been removed. In such cases, the AP node is deleted from the graph, effectively removing its presence from the fingerprint database.

After the APs changes are recorded by edge changes in the graph, using the new edges and the new node features, we retrain the feature extractor according to the same procedure as mentioned before (Section 4.1). The feature vectors are again fed to the GNN for inference (Section 5.1). The output embeddings are then used by the decoder in the feature extractor to obtain the updated RSS values (Section 5.2).

7 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate GUFU. We first present our experiment settings in Section 7.1 and then compare the performance of GUFU with state-of-the-art algorithms in Section 7.2. We further demonstrate the effectiveness of each system component of GUFU and the impact of the system parameters in Section 7.3.

7.1 Experimental Setup

Data collection: We conduct experiments on four different sites, including one campus building (Campus), two shopping malls (Malls A and B), and one hospital (Hospital), as shown in Figure 8. The campus building has three floors, and each of the other sites has four floors. For each floor in the buildings, we construct an initial

---

Algorithm 2: GUFU: Edge Modification

Input: Graph $\mathcal{G}(V, E \cup E_{virtual}, W)$; extracted node feature $\mathbf{z}_v$, $\forall v \in V$.

Output: New edges E_n and corresponding edge weights W_n.

1 $\overline{w}_{uv} = \frac{w_{uv}}{w_{\max}}, \forall e_{uv} \in E \cup E_{virtual}$.

2 $G^{(0)}(v) = F^{(0)}(v) = \mathbf{z}_v, \forall v \in V$.

3 $E_n = W_n = \{\}$

4 $t = 0$.

5 **do**

    /* Iteratively update G and F.   */

6   **for** $v \in V$ **do**

        for $i \in [1, d]$ **do**

7              $g_i^{(t+1)}(v) = \frac{1}{|N(v)|} \sum_{u \in N(v)} f_i^{(t)}(u) \overline{w}_{uv}$

8              $f_i^{(t+1)}(v) = 1 - \frac{1}{2|N(v)|} \sum_{u \in N(v)} |\overline{w}_{uv} - g_i^{(t+1)}(u)|$

        end

    end

9      $t = t + 1$.

    while $\sum_{v \in V} \|F^{(t)}(v) - F^{(t-1)}(v)\|_2 > \epsilon$ and $\sum_{v \in V} \|G^{(t)}(v) - G^{(t-1)}(v)\|_2 > \epsilon$;

10 **for** $s, u, v \in V : e_{su} \in E, e_{uv} \in E_{virtual}$ **do**

    /* s is connected to u but not necessarily v. u and v are virtual neighbors.   */

11   **if** $v \notin N(s)$ **then**

12          $\hat{w}_{sv} = \frac{1}{2} w_{\max} (G(s) \cdot F(v) + F(s) \cdot G(v))$

13   **if** $\hat{w}_{sv} \geq \delta$ and $e_{sv} \notin E$ **then**

            /* Add an edge between s and v.   */

14              $E_n = E_n \cup \{e_{sv}\}$

15              $W_n = W_n \cup \{\hat{w}_{sv}\}$

        end

16   **if** $\hat{w}_{sv} < \delta$ and $e_{sv} \in E$ **then**

            /* Remove the edge between s and v.   */

17              $E_n = E_n \setminus \{e_{sv}\}$

18              $W_n = W_n \setminus \{w_{sv}\}$

        end

    end

    end

19 **return** E_n, W_n.

---

fingerprint database by dividing the site into grids and collecting WiFi signals from the center of each grid. After that, WiFi signals are crowdsourced periodically for updates. For Campus and Mall A, we collect data once per week and once every two weeks, respectively, for eight weeks. For Mall B and Hospital, the data are collected every week for four weeks. For reliable performance evaluation, the measurement locations of the collected signals are also recorded. As a result, those locations can be used as ground truth, yet only for measuring the prediction error. In Table 3, we provide the details of the data collection process for fingerprint updates in four buildings. Additionally, to provide a clearer overview of the number of signal samples collected and the changes in the number of APs within each batch of new data, we present relevant statistics in Table 4 and Table 5.

Table 3. Data collection details in different sites.

Site	Area (m^2)	Grid size (m^2)	Duration (weeks)	Frequency (/week)
Campus	255×95	2.5×2.5	8	1
Mall A	180×105	2.5×2.5	8	0.5
Mall B	650×280	2.0×2.0	4	1
Hospital	200×120	2.0×2.0	4	1

Table 4. Number of samples collected on each floor for all sites.

Site	Floor	Init	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8
Campus	GF	11284	4029	4236	4661	4264	4342	4515	4509	4190
	1F	10037	3589	3498	3981	3821	3445	3781	3448	3968
	2F	11256	3733	3680	3667	3873	4110	3947	3899	3665
Mall A	B1F	6284	-	2544	-	1788	-	2056	-	2272
	GF	7619	-	2487	-	2338	-	2843	-	2755
	1F	3326	-	449	-	621	-	1054	-	965
	2F	4158	-	1642	-	1153	-	1609	-	1629
Mall B	1F	131586	45417	46146	48439	46392	-	-	-	-
	MF	49453	16820	16394	16635	15660	-	-	-	-
	2F	62882	21286	20567	20396	19944	-	-	-	-
	3F	57194	52427	50251	53389	54803	-	-	-	-
Hospital	GF	6154	2078	1733	1745	3936	-	-	-	-
	1F	4318	258	419	419	1448	-	-	-	-
	2F	2202	422	731	732	2305	-	-	-	-
	4F	1931	194	194	118	263	-	-	-	-

Table 5. Summary of AP changes on each floor for all sites. +/- stands for addition/removal.

| Site	Floor	Init	Week1 +/-	Week2 +/-	Week3 +/-	Week4 +/-	Week5 +/-	Week6 +/-	Week7 +/-	Week8 +/-	Total +/-
Campus	GF	391	53/46	33/26	29/22	3/2	41/6	17/28	25/23	3/4	204/136
	1F	306	55/32	21/4	15/13	6/2	41/8	20/16	20/19	3/6	180/100
	2F	253	10/7	19/5	12/9	0/4	27/3	5/8	11/7	6/13	90/56
Mall A	B1F	927	-	17/10	-	43/27	-	58/43	-	55/21	140/68
	GF	1328	-	57/48	-	31/77	-	29/53	-	41/61	104/184
	1F	434	-	18/24	-	37/38	-	24/37	-	11/34	47/90
	2F	385	-	14/21	-	30/33	-	35/31	-	24/28	140/68
Mall B	1F	494	63/41	18/52	15/27	6/4	-	-	-	-	82/104
	MF	199	3/1	5/4	2/2	0/3	-	-	-	-	8/8
	2F	323	54/20	29/39	37/22	16/20	-	-	-	-	121/90
	3F	304	18/30	33/62	50/39	14/13	-	-	-	-	64/93
Hospital	GF	673	8/4	5/8	5/0	4/1	-	-	-	-	17/8
	1F	317	0/0	3/10	7/6	10/1	-	-	-	-	18/15
	2F	266	0/0	11/7	2/0	3/0	-	-	-	-	15/6
	4F	164	0/0	1/1	3/2	0/0	-	-	-	-	4/3
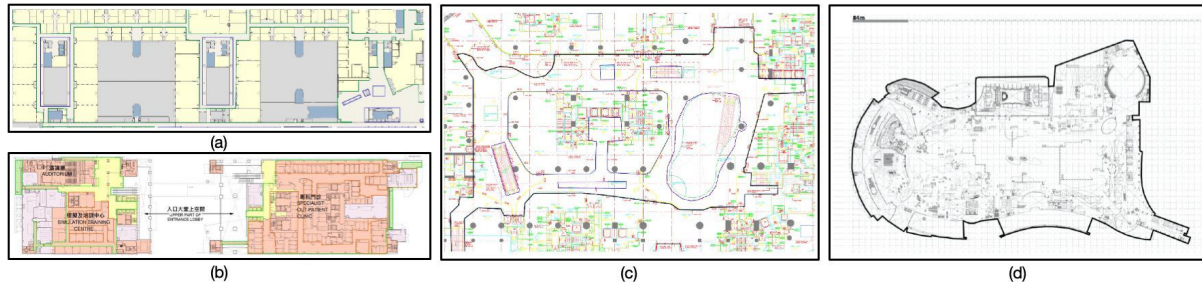
Fig. 8. Floor plans of one floor from different sites. (a) Campus. (b) Hospital. (c) Mall A. (d) Mall B.

Implementation details: In GUFU, for the autoencoder in feature extraction, we use a dropout rate of 0.5. The encoder has its input dimension as the number of MACs detected in the initial fingerprint database and its output dimension of 32. For the GNN in GUFU, we apply a learning rate of 0.01 and a dropout of 0.5. The model is trained for 50 epochs. We use ReLU as the activation function ϕ. We set $\sigma = 0.95$ for the threshold value for creating virtual edges, $\alpha = 0.5$ in Equation (9), and $\epsilon = 0.1$ for the link prediction threshold. The code is available anonymously online.[1]

State-of-the-art algorithms: To evaluate the performance of GUFU on the fingerprint updates, we compare its performance with four state-of-the-art algorithms, namely, Fidora [39], iToLoc [20], MTDAN [37] and WiDAGCN [47]. Each algorithm is briefly summarized as follows:

- Fidora [39]: It uses a classification neural network and a reconstruction neural network to infer location labels for new signals and update RSS values in the original fingerprints via semi-supervised learning. It is designed based on the assumption that the signal characteristics, e.g., RSS values and detected APs, are different over different areas, which are grids in our experiments.

- iToLoc [20]: It is based on the assumption that there are temporal features of WiFi signals that are consistent over time. Hence, it first models WiFi signals as a 2D image and extracts time-invariant features from the signals at different times using a convolutional time discriminator. It then predicts the locations of new signals using another convolutional neural network. Both networks are trained from the fingerprint database and the first batch of crowdsourced signals.

- MTDAN [37]: Similar to our GUFU, MTDAN's update is also done with the assumption that there are AP differences in signal samples from different times. Utilizing multi-target domain adaptation to extract time-invariant signal features from stable APs (shared APs in our context), MTDAN is able to predict location labels for new signal samples.

- WiDAGCN [47]: It is one of the state-of-the-art graph modeling for signal fingerprint update. By modeling APs and signal samples as different nodes in the graph and using graph attention, WiDAGCN can match graphs constructed from new signals to subgraphs from the existing graphs, and thus get its signal and location predictions.

Note that iToLoc [20] does not update the RSS values in the fingerprint database. For a fair comparison, we use our thresholding method (for creating virtual edges) in GUFU to do so, along with iToLoc. Specifically, for each signal record/sample in the database, which appears as a sample node in the graph, we first find its closest neighbors, which are also sample nodes in the graph, with their feature cosine similarity higher than $\sigma = 0.95$. Then, whenever the location of a new signal sample is predicted by iToLoc, its corresponding signal record/sample

---

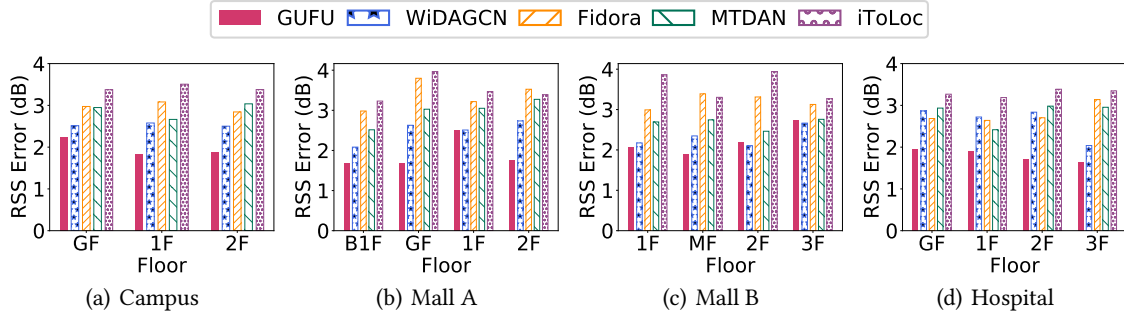[1]Code implementation: https://github.com/khchiuac/GUFU

Fig. 9. Summary of RSS error on different floors of the four sites.

Table 6. Summary of the mean (standard deviation) for location prediction errors in different sites.

Site	GUFU	Fidora	WiDAGCN	MTDAN	iToLoc
Campus	**4.92m** (0.815m)	5.48m (0.781m)	5.67m (0.919m)	6.22m (0.738m)	6.53m (0.833m)
Mall A	**4.44m** (0.974m)	4.88m (1.010m)	5.06m (1.071m)	5.21m (1.268m)	5.49m (1.541m)
Mall B	**4.39m** (1.252m)	5.46m (1.872m)	5.39m (1.997m)	5.77m (1.825m)	5.94m (1.808m)
Hospital	**3.06m** (0.603m)	3.72m (0.466m)	4.03m (0.704m)	3.99m (0.840m)	4.18m (0.597m)

is found in the database, and each RSS value in the record is updated as the weighted average of the RSS values from its neighboring records/samples (i.e., its neighbors in the graph).

Evaluation metrics: To measure the accuracy of location prediction, we use the 'location error' that is defined as the average Euclidean distance between the predicted locations and their ground-truth locations, as widely used in the literature. In addition, to measure how accurately the fingerprint database has been updated, we use the 'RSS error' that is defined as the average difference between updated RSS values and actually measured RSS values of APs. All the experiments are done on a machine with an Intel(R) Core(TM) i9-9900X CPU @ 3.50GHz, 64G RAM and two graphic cards of Nvidia GeForce RTX 2080 Ti.

7.2 Overall Performance

Fingerprint updates: We first evaluate the performance of GUFU and other baseline algorithms for updating *aged* fingerprints, which is measured in the RSS error. As shown in Figure 9, GUFU outperforms the other algorithms substantially over all four sites.[2] It indicates the effectiveness of our representation learning for WiFi signal samples and the fingerprint-updating MLP networks built upon the representations. Fidora, however, does not perform well because signal characteristics in neighboring areas can be similar to each other, which is in contrast to the rationale behind its design. The performance of iToLoc, MTDAN, and WiDAGCN is also not satisfactory. While they update RSS values based on the predicted locations of new signals, their location predictions are not as accurate as GUFU. See Figures 10–13 for more details on the location prediction accuracy, which shall be explained below. Furthermore, all the baseline algorithms do not consider newly introduced APs in the new signals, thereby negatively affecting the location prediction accuracy and the quality of fingerprint updates.

Location prediction accuracy: We summarize the average location prediction errors of GUFU and other state-of-the-art algorithms in Table 6. GUFU outperforms other algorithms significantly, with up to 82% improvement

---

[2]A decrease of 3dB in signal strength indicates that the signal power reduces by half.
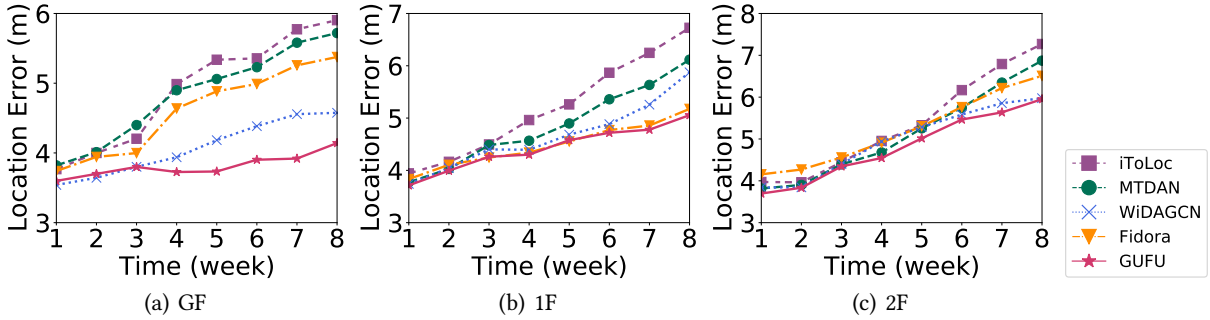
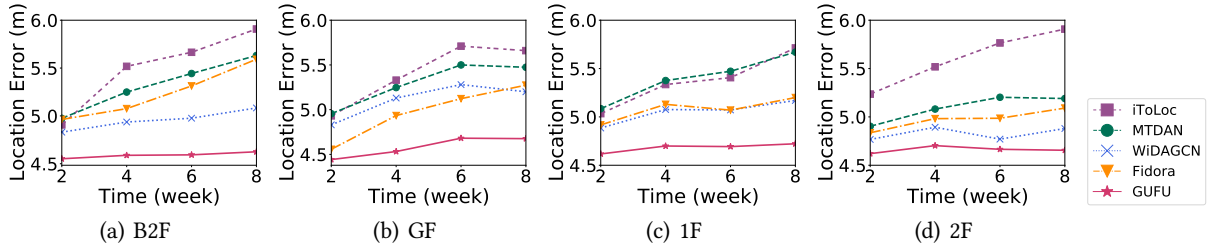Fig. 10. Location error over eight weeks on three floors in the campus.

Fig. 11. Location error over eight weeks on four floors in mall A.

in the location error. This is because GUFU accurately updates fingerprints over time thanks to our novel graph-based representation learning for WiFi signals along with effective link predictions. However, iToLoc and MTDAN do not perform well as their signal features may vary over time, which affects the performance of their time discriminators. Fidora is also not able to correctly classify new signals (or infer their locations) since possibly similar signal characteristics over the neighboring areas may hinder its discriminative power. WiDAGCN does not perform well because its subgraph matching and prediction may be based on outdated fingerprint information. Moreover, they do not include newly added APs in the fingerprints, which affects their performance over time.

We further evaluate the accuracy of location prediction week by week for newly collected signals in all sites and present the results in Figures 10–13. GUFU's performance varies at different floors as different floors may exhibit different RF signal environments due to diverse factors such as distinct floor designs, different densities of APs, and their different power levels. Nonetheless, GUFU consistently outperforms other algorithms over the *whole* time period. The improvement of GUFU over the other ones becomes more and more significant as time goes by. This indicates that using GUFU, the fingerprints can be valid for a longer period of time, and a recollection of signals for fingerprint database construction can be done less often. In addition, we show the CDF of location prediction errors for four sites in Figure 14. While the prediction errors of GUFU are mostly within 6–8m, the others lead to larger errors. For example, the 90-th percentile accuracy of GUFU outperforms WiDAGCN, Fidora, MTDAN, and iToLoc by up to 31.6%, 38.7%, 39.6%, and 40.4%, respectively. All these results validate the effectiveness of GUFU in location prediction and demonstrate its superior performance to the other baseline algorithms.
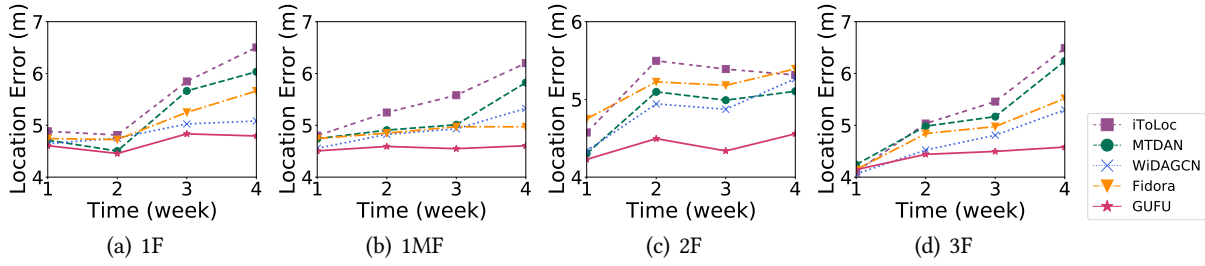
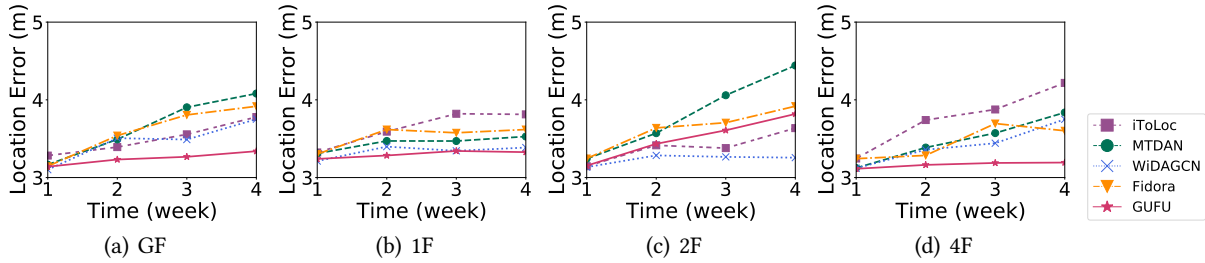Fig. 12. Location error over eight weeks on four floors in mall B.

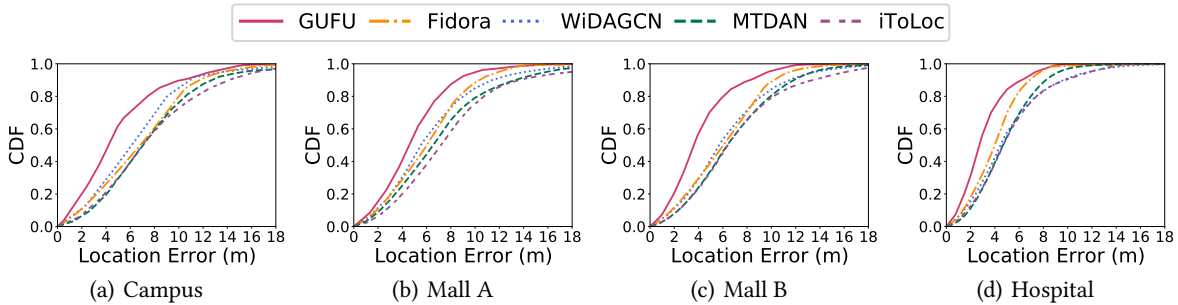Fig. 13. Location error over eight weeks on four floors in the hospital.

Fig. 14. Comparison of CDF on location error for four different sites.

7.3 Ablation Study

Time required for each update: We recorded the time required for each update and report the results in Table 7. The results show that GUFU's update time for each batch of new data is similar to the existing models like iToLoc and MTDAN, while being significantly faster than Fidora and WiDAGCN.

Impact of RSS-feature extractor: We use an autoencoder as the feature extractor to learn the information carried by the fingerprints. The rationale behind our choice of the autoencoder is that it has the capability to "reconstruct" signals from the extracted signal features using its decoder. This structure aligns well with our system design, where the extracted signal features are intended to predict the signal strengths needed for fingerprint updates. To show the benefits of using such a feature extractor, we quantitatively compare the location error and RSS error with and without this module in Figure 15(a)–(b). For GUFU without feature extraction, we directly use the normalized RSS values as the node features on the graph, i.e., adding an offset of 120dBm and then dividing the obtained value by 120. As shown in the plots, GUFU achieves a substantial improvement over

Table 7. Time needed (in seconds) for each update over eight weeks on the campus.

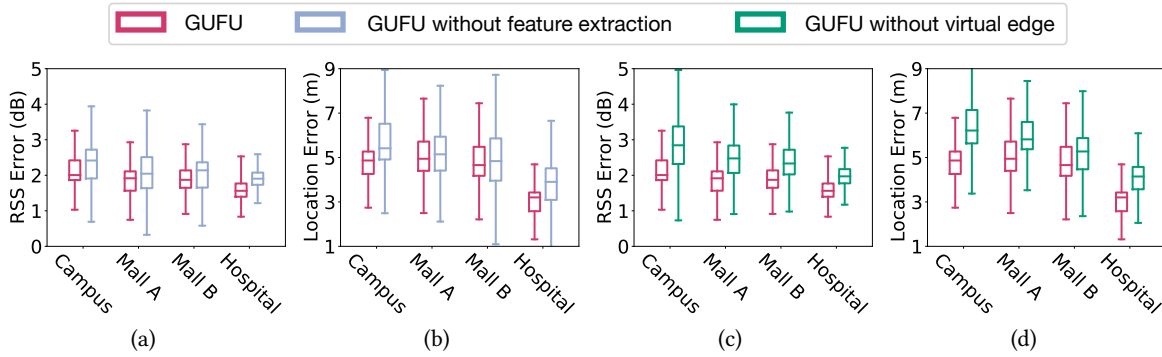| Method	Init	Update1	Update2	Update3	Update4	Update5	Update6	Update7	Update8
GUFU	561.64	94.19	96.63	99.11	91.24	94.68	92.65	98.84	95.10
Fidora	692.81	102.18	109.49	115.39	106.16	108.97	102.56	102.28	99.30
iToLoc	464.49	86.36	81.93	85.05	88.62	76.79	75.40	74.72	89.08
MTDAN	537.78	92.28	93.32	96.83	93.64	97.43	92.46	95.22	93.29
WiDAGCN	987.30	149.70	152.65	169.33	158.86	161.31	165.36	163.81	156.94

Fig. 15. Ablation study of GUFU. (a) and (b) GUFU (without feature extraction); (c) and (d) GUFU (without virtual edges).

Table 8. Average training time (in seconds) for GUFU to converge using different initialization schemes.

Initialization	Campus	Mall A	Mall B	Hospital
Weighted Average	561.64	193.41	1205.36	104.29
Zero Initialization	813.72	269.73	1807.44	130.16
One Initialization	801.231	295.72	1755.39	128.51
Random Initialization	603.20	199.64	1373.89	105.59

its version without the feature extractor in RSS error and location error, by up to 26.6% and 18.9%, respectively. This demonstrates that the initialization of node features with the feature extractor indeed helps to better capture similarities between WiFi signal samples and thus improves the location prediction and fingerprint updating results.

We also observe that the autoencoder extracts signal features that *most effectively* represent the original signal strengths, as shown in Figure 16. We compare the autoencoder against other popular methods for obtaining fixed-size, lower-dimensional features from a longer list of input signal strengths. Specifically, we consider statistical methods such as principal component analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE), and uniform manifold approximation and projection (UMAP), and learning-based ones such as multilayer perceptrons (MLPs) and convolutional neural networks (CNNs). Here we do not consider more complex models such as generative adversarial networks and vision transformers, because our model needs to be retrained to adapt to changes in the dimensions of input signal strengths resulting from modifications in APs. As shown in Figure 16, GUFU having the autoencoder as the feature extractor achieves the *best* performance.

Choice of initialization schemes for node features: For graph initialization, our purpose is to initialize the node features of both sampled nodes and AP nodes in a way that reflects their proximity relationships in physical
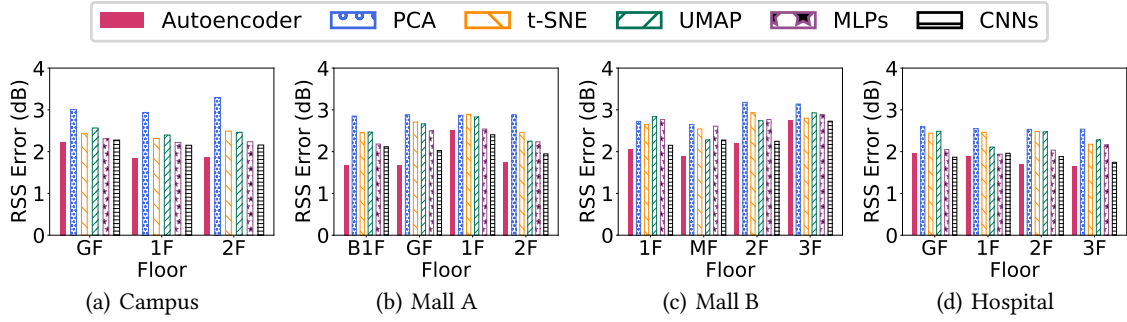
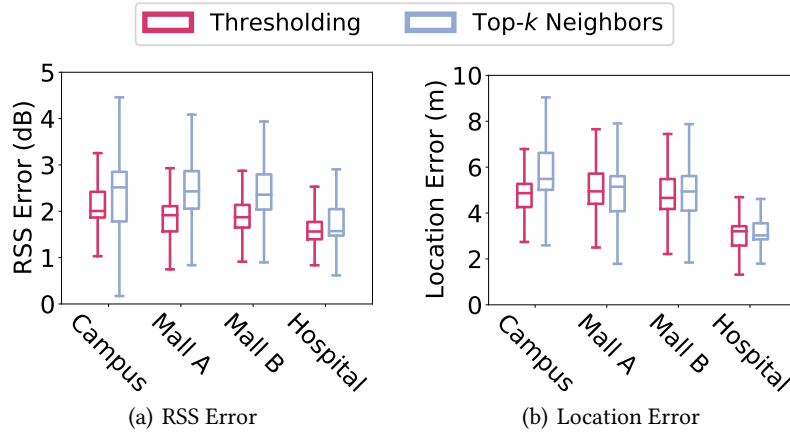Fig. 16. Summary of RSS error on different floors of the four sites using different feature extractors.

Fig. 17. Thresholding ($\sigma = 0.95$) versus top-k (k=10) neighbors in virtual edge creation.

space. Thus, for the sample nodes, we initialize their feature vectors using the output from the feature extractor, which takes their normalized RSS values from the APs as input. For each AP node, we initialize its feature vector as a weighted average of the feature vectors of its connected sample nodes (i.e., those in its immediate neighborhood), with weights being proportional to the edge weights that are (shifted) RSS values from the AP. That is, the feature vector of each AP is initialized as the result of an information aggregation from its connected sample nodes while also reflecting their proximity relationships in physical space (according to the RSS values).

We have also examined the impact of different initializations on the training time of our GNN model. Specifically, in addition to our weighted averaging, we consider (1) zero-initialization, where each entry is set to 0, (2) one-initialization, where each entry is set to 1, and (3) random initialization, where each entry is assigned a value that is chosen uniformly at random from [0, 1]. As shown in Table 8, our weighted averaging achieves the *fastest* training time, which supports the rationale behind our weighted averaging.

Impact of virtual edges: The virtual edges enable direct interaction between sample nodes with similar signal features. To validate the effectiveness of this design, we compare GUFU's performance with and without virtual edges in Figure 15(c)–(d). Including the virtual edges reduces the RSS error and the location error by 11.8% and 15.2%, respectively. The results indicate that virtual edges help to propagate useful information between similar sample nodes such that node embeddings can be better learned for location prediction and signal updates.
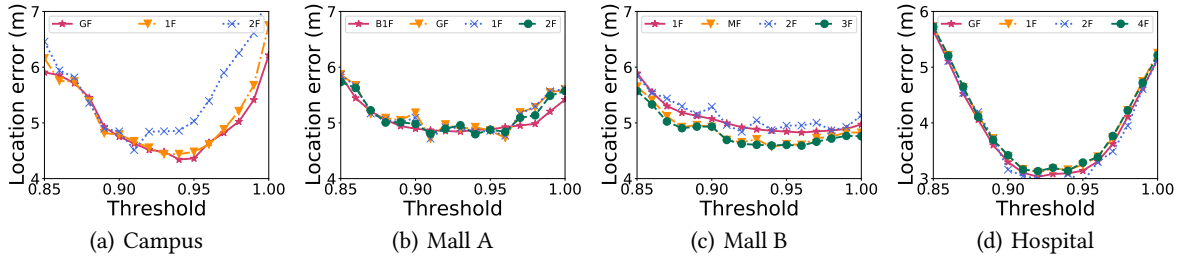
Fig. 18. Location errors on different floors of the four sites using different threshold values of σ in virtual edge creation.
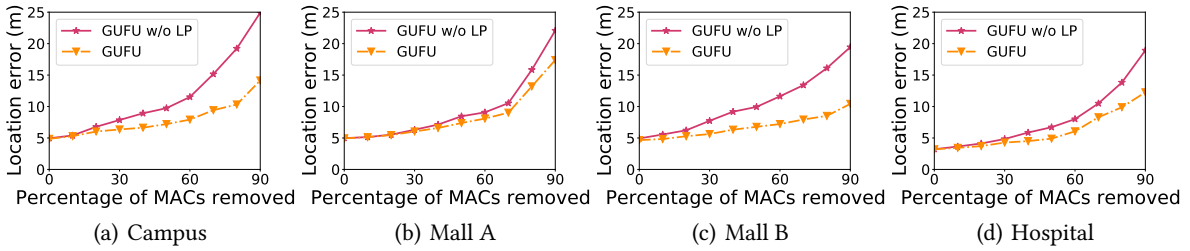

Fig. 19. Location error versus the ratio of MACs removed in four sites.

In addition, our virtual edges are created based on the thresholding of cosine similarity values between signal features of two sample nodes. We compare it with creating virtual edges using top-k (k=10) nearest neighbors, and show the performance of the two schemes in Figure 17. Our thresholding-based method outperforms top-k nearest neighbors substantially, by 16.2% in RSS error and 22.1% in location error, respectively. This is because, in top-k nearest neighbors, there may be sample nodes with low feature similarities selected, as illustrated in Figure 5, which degrades the performance of location prediction and signal updates. With thresholding, the high feature similarity between sample nodes with virtual edges can be guaranteed, leading to high accuracy in location prediction and signal updates.

In GUFU, the value of threshold σ was determined by a hyper-parameter tuning process. We examined the performance of GUFU with varying threshold values and now present the results in Figure 18. We observed that threshold values between 0.90 and 0.97 yield good performance for GUFU's location prediction. For simplicity, we set $\sigma = 0.95$ for all sites.

Impact of link prediction: Link prediction enriches the fingerprint database with new connections between AP nodes and sample nodes. To show how much it improves GUFU's performance, we intentionally remove AP nodes at random in the original fingerprint database. As a result, these MACs will become the newly detected ones in the updating phase. We compare the performance of GUFU and GUFU without link prediction (GUFU w/o LP) in Figure 19. With more MACs removed from the original fingerprint database, the performance of both schemes degrades. However, compared to GUFU without link prediction, the errors of GUFU increase slowly. This is because GUFU can retain more AP information over time with the link prediction capability. In particular, with up to 90% of randomly selected MAC addresses removed, GUFU still has an average location error of 15.36m, which is 46.2% better than that of GUFU without link prediction.

Impact of the proportion of evolved APs: GUFU is designed to handle both the addition and removal of APs. To further demonstrate the model's robustness against varying proportions of new and removed APs, we sampled these APs from the campus site, which experienced the most significant AP changes among the four sites in
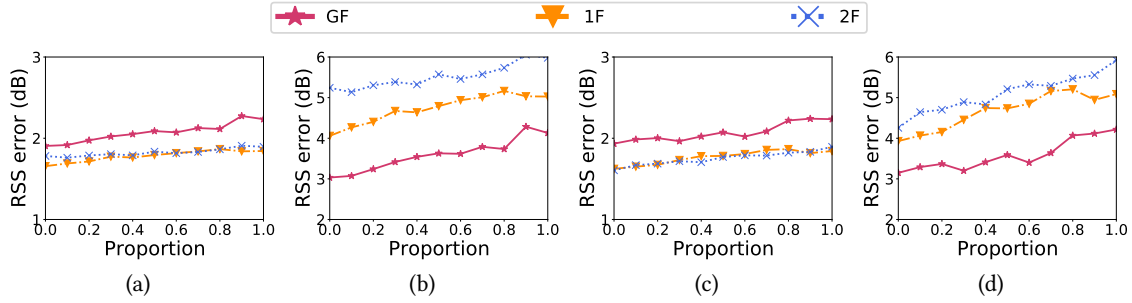
Fig. 20. GUFU's performance over different AP settings. (a) and (b) new APs. (c) and (d) removed APs.

our experiment. In Figure 20, we show GUFU's performance with varying proportions of added and removed APs. Figure 20(a) and Figure 20(b) show GUFU's performance considering all removed APs alongside different proportions of new APs, while Figure 20(c) and Figure 20(d) depict the model's performance with all new APs and varying proportions of removed APs. The results indicate that GUFU's performance remains relatively consistent across different proportions of evolved APs.

It is noteworthy that while GUFU's performance remains relatively stable across varying proportions of new and removed APs, it tends to perform better with smaller proportions of evolved APs. Consequently, we anticipate that GUFU may reach a limit where the signal prediction error exceeds a certain threshold, necessitating periodic signal recollections. Nonetheless, as shown in Section 7.2, the fingerprint database updated using GUFU deteriorates more slowly than those maintained by other state-of-the-art methods. As a result, the time interval between consecutive collections can be longer compared to fingerprints managed by those methods, potentially extending beyond several years. We leave the determination of the exact time interval needed between these consecutive recollections as a future work.

8 CONCLUSION

In this paper, we propose GUFU, an effective graph-based approach for crowdsourced WiFi fingerprint updates using unlabeled WiFi signals. Our approach relies solely on RSS values from ambient access points (APs) on the site, without requiring any additional information. When a batch of newly collected, yet unlabeled WiFi signals becomes available, GUFU leverages this data to update the existing fingerprints. It is designed to adapt to changes in RSS values, the presence of new APs, and the potential removal of APs in the environment over time. To validate the effectiveness of GUFU, we developed a prototype and conducted extensive evaluations across four different sites over a significant period, ranging from one month to eight months. The experimental results demonstrate that GUFU surpasses other state-of-the-art algorithms in both location prediction and fingerprint updates. Overall, our findings showcase the potential of GUFU in enabling the long-term deployment of a fingerprint database and its automatic enhancement over time, leading to improved fingerprinting-based services.

References

[1] Moustafa Abbas, Moustafa Elhamshary, Hamada Rizk, Marwan Torki, and Moustafa Youssef. 2019. WiDeep: WiFi-based Accurate and Robust Indoor Localization System using Deep Learning. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

[2] Pan Chen and Shuiping Zhang. 2024. DeepMetricFi: Improving Wi-Fi Fingerprinting Localization by Deep Metric Learning. *IEEE Internet of Things Journal* 11, 4 (2024), 6961–6971. https://doi.org/10.1109/JIOT.2023.3315289

[3] Xi Chen, Hang Li, Chenyi Zhou, Xue Liu, Di Wu, and Gregory Dudek. 2020. Fido: Ubiquitous fine-grained wifi-based localization for unlabelled users via domain adaptation. In *Proceedings of The Web Conference 2020*.

[4] Zhiwei Chen, Zhibo Pang, Wenjing Hou, Hong Wen, Mi Wen, Runhui Zhao, and Tao Tang. 2024. Cross-Device Radio Frequency Fingerprinting Identification Based On Domain Adaptation. *IEEE Transactions on Consumer Electronics* (2024), 1–1. https://doi.org/10.1109/TCE.2024.3357844

[5] Jim Cherian, Jun Luo, and Shen-Shyang Ho. 2018. ParkLoc: Light-weight Graph-based Vehicular Localization in Parking Garages. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3, Article 99 (sep 2018), 23 pages. https://doi.org/10.1145/3264909

[6] Jean-François Determe, Sophia Azzagnuni, Utkarsh Singh, François Horlin, and Philippe De Doncker. 2022. Monitoring Large Crowds With WiFi: A Privacy-Preserving Approach. *IEEE Systems Journal* 16, 2 (2022), 2148–2159.

[7] Yinhuan Dong, Tughrul Arslan, and Yunjie Yang. 2022. An Encoded LSTM Network Model for WiFi-based Indoor Positioning. In *2022 IEEE 12th International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. 1–6. https://doi.org/10.1109/IPIN54987.2022.9918116

[8] Shukai Fan, Yongzhi Wu, Chong Han, and Xudong Wang. 2021. Siabr: A structured intra-attention bidirectional recurrent deep learning method for ultra-accurate terahertz indoor localization. *IEEE J. Sel. Areas Commun.* (2021).

[9] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research* 17, 1 (2016), 2096–2030.

[10] Zhihui Gao, Yunfan Gao, Sulei Wang, Dan Li, and Yuedong Xu. 2020. CRISLoc: Reconstructable CSI fingerprinting for indoor smartphone localization. *IEEE Internet of Things Journal* 8, 5 (2020), 3422–3437.

[11] Yao Ge, Ahmad Taha, S.A Shah, Kia Dashtipour, Shuyuan Zhu, Jonathan M. Cooper, Qammer Abbasi, and Muhammad Imran. 2022. Contactless WiFi Sensing and Monitoring for Future Healthcare - Emerging Trends, Challenges and Opportunities. *IEEE Reviews in Biomedical Engineering* (2022).

[12] Baoshen Guo, Weijian Zuo, Shuai Wang, Wenjun Lyu, Zhiqing Hong, Yi Ding, Tian He, and Desheng Zhang. 2022. WePos: Weak-supervised Indoor Positioning with Unlabeled WiFi for On-demand Delivery. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 2, Article 54 (jul 2022), 25 pages. https://doi.org/10.1145/3534574

[13] Lingchao Guo, Zhaoming Lu, Shuang Zhou, Xiangming Wen, and Zhihong He. 2021. Emergency Semantic Feature Vector Extraction From WiFi Signals for In-Home Monitoring of Elderly. *IEEE Journal of Selected Topics in Signal Processing* 15, 6 (2021), 1423–1438. https://doi.org/10.1109/JSTSP.2021.3109429

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[15] Suining He, Wenbin Lin, and S.-H. Gary Chan. 2017. Indoor Localization and Automatic Fingerprint Update with Altered AP Signals. *IEEE Transactions on Mobile Computing* 16, 7 (2017), 1897–1910.

[16] Rajalaxmi Hegde, Sandeep Kumar Hegde, Kdv Prasad, Ved Srinivas, Tanmoy De, and V Dankan Gowda. 2023. Wi-Fi Router Signal Coverage Position Prediction System using Machine Learning Algorithms. In *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)*. 253–258. https://doi.org/10.1109/ICSCSS57650.2023.10169501

[17] Baoqi Huang, Zhendong Xu, Bing Jia, and Guoqiang Mao. 2019. An Online Radio Map Update Scheme for WiFi Fingerprint-Based Localization. *IEEE Internet of Things Journal* 6, 4 (2019), 6909–6918.

[18] Gang Huang, Zhaozheng Hu, Jie Wu, Hanbiao Xiao, and Fan Zhang. 2020. WiFi and vision-integrated fingerprint for smartphone-based self-localization in public indoor scenes. *IEEE Internet of Things Journal* 7, 8 (2020), 6748–6761.

[19] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 221–230.

[20] Danyang Li, Jingao Xu, Zheng Yang, and Chengpei Tang. 2024. Train Once, Locate Anytime for Anyone: Adversarial Learning-based Wireless Localization. *ACM Trans. Sen. Netw.* 20, 2, Article 37 (jan 2024), 21 pages. https://doi.org/10.1145/3614095

[21] Hualiang Li, Zhihong Qian, Guiqi Liu, and Xue Wang. 2022. NQRELoc: AP Selection via Nonuniform Quantization RSSI Entropy for Indoor Localization. *IEEE Sensors Journal* 22, 10 (2022), 9724–9732. https://doi.org/10.1109/JSEN.2022.3166072

[22] Wei Li, Cheng Zhang, and Yoshiaki Tanaka. 2020. Pseudo Label-Driven Federated Learning-Based Decentralized Indoor Localization via Mobile Crowdsourcing. *IEEE Sensors Journal* 20, 19 (2020), 11556–11565.

[23] Yiming Lin, Daokun Jiang, Roberto Yus, Georgios Bouloukakis, Andrew Chio, Sharad Mehrotra, and Nalini Venkatasubramanian. 2020. Locater. *Proceedings of the VLDB Endowment* 14, 3 (nov 2020), 329–341. https://doi.org/10.14778/3430915.3430923

[24] Yiming Lin, Pramod Khargonekar, Sharad Mehrotra, and Nalini Venkatasubramanian. 2021. T-Cove: An Exposure Tracing System Based on Cleaning Wi-Fi Events on Organizational Premises. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2783–2786. https://doi.org/10.14778/3476311.3476344

[25] An Liu, Zhe Huang, Min Li, Yubo Wan, Wenrui Li, Tony Xiao Han, Chenchen Liu, Rui Du, Danny Kai Pin Tan, Jianmin Lu, et al. 2022. A survey on fundamental limits of integrated sensing and communication. *IEEE Communications Surveys & Tutorials* (2022).

[26] Chris Xiaoxuan Lu, Yuanbo Xiangli, Peijun Zhao, Changhao Chen, Niki Trigoni, and Andrew Markham. 2019. Autonomous Learning of Speaker Identity and WiFi Geofence From Noisy Sensor Data. *IEEE Internet of Things Journal* 6, 5 (2019), 8284–8295. https://doi.org/10.1109/JIOT.2019.2926645

[27] Sumudu Hasala Marakkalage, Billy Pik Lik Lau, Yuren Zhou, Ran Liu, Chau Yuen, Wei Quin Yow, and Keng Hua Chong. 2021. WiFi Fingerprint Clustering for Urban Mobility Analysis. *IEEE Access* 9 (2021), 69527–69538. https://doi.org/10.1109/ACCESS.2021.3077583

[28] Dang Pham-Hai, Ninh Duong-Bao, Jing He, Luong Nguyen Thi, Seon-Woo Lee, and Khanh Nguyen-Huu. 2023. WiFi-based Positioning System with k-means Clustering and Outlier Removal: Evidence from Multiple Datasets. In *Proceedings of the 12th International Symposium on Information and Communication Technology* (, Ho Chi Minh, Vietnam,) *(SOICT '23)*. Association for Computing Machinery, New York, NY, USA, 980–988. https://doi.org/10.1145/3628797.3628920

[29] Abubakarsidiq Makame Rajab and Bang Wang. 2023. IWFUCIA: An integrated model for Wi-Fi fingerprints radio map updating and localization on changed impact analysis. *Pervasive and Mobile Computing* 91 (2023), 101784. https://doi.org/10.1016/j.pmcj.2023.101784

[30] Nader G. Rihan, Mahmoud Abdelaziz, and Samy S. Soliman. 2022. A Hybrid Deep-learning/Fingerprinting for Indoor Positioning Based on IEEE P802.11az. *2022 5th International Conference on Communications, Signal Processing, and their Applications (ICCSPA)* (2022), 1–6.

[31] Peter Sarcevic, Dominik Csik, and Akos Odry. 2023. Indoor 2D Positioning Method for Mobile Robots Based on the Fusion of RSSI and Magnetometer Fingerprints. *Sensors* 23, 4 (2023). https://doi.org/10.3390/s23041855

[32] Shuang Shang and Lixing Wang. 2022. Overview of WiFi fingerprinting-based indoor positioning. *IET Communications* 16, 7 (2022), 725–733.

[33] Xudong Song, Xiaochen Fan, Xiangjian He, Chaocan Xiang, Qianwen Ye, Xiang Huang, Gengfa Fang, Liming Luke Chen, Jing Qin, and Zumin Wang. 2019. CNNLoc: Deep-Learning Based Indoor Localization with WiFi Fingerprinting. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 589–595.

[34] Yu Tian, Jiankun Wang, and Zenghua Zhao. 2021. Wi-Fi Fingerprint Update for Indoor Localization via Domain Adaptation. In *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*. 835–842.

[35] Amee Trivedi, Camellia Zakaria, Rajesh Balan, Ann Becker, George Corey, and Prashant Shenoy. 2021. WiFiTrace: Network-based Contact Tracing for Infectious Diseases Using Passive WiFi Sensing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 1, Article 37 (mar 2021), 26 pages. https://doi.org/10.1145/3448084

[36] Cristian Turetta, Florenc Demrozi, Philipp H. Kindt, Alejandro Masrur, and Graziano Pravadelli. 2022. Practical identity recognition using WiFi's Channel State Information. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 76–79. https://doi.org/10.23919/DATE54114.2022.9774772

[37] Jiankun Wang, Zenghua Zhao, Mengling Ou, Jiayang Cui, and Bin Wu. 2023. Automatic Update for Wi-Fi Fingerprinting Indoor Localization via Multi-Target Domain Adaptation. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 2, Article 78 (jun 2023), 27 pages. https://doi.org/10.1145/3596239

[38] Xijia Wei, Zhiqiang Wei, and Valentin Radu. 2021. MM-Loc: Cross-sensor indoor smartphone location tracking using multimodal deep neural networks. In *2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 1–8.

[39] X. Chen, H. Li, C. Zhou, X. Liu, D. Wu, and G. Dudek. 2022. Fidora: Robust WiFi-based Indoor Localization via Unsupervised Domain Adaptation. *IEEE Internet of Things Journal* (2022).

[40] Chenlu Xiang, Shunqing Zhang, Shugong Xu, and George C Alexandropoulos. 2021. Self-calibrating indoor localization with crowd-sourcing fingerprints and transfer learning. In *ICC 2021-IEEE International Conference on Communications*. IEEE, 1–6.

[41] Han Xu, Zheng Yang, Zimu Zhou, Longfei Shangguan, Ke Yi, and Yunhao Liu. 2015. Enhancing wifi-based localization with visual clues. In *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*.

[42] Lyuxiao Yang, Nan Wu, Bin Li, Weijie Yuan, and Lajos Hanzo. 2022. Indoor Localization Based on Factor Graphs: A Unified Framework. *IEEE Internet of Things Journal* (2022).

[43] Sheng Yang, Jingbin Liu, Xiaodong Gong, Gege Huang, and Yu Bai. 2021. A robust heading estimation solution for smartphone multisensor-integrated indoor positioning. *IEEE Internet of Things Journal* 8, 23 (2021), 17186–17198.

[44] Xi Yang, Suining He, Bing Wang, and Mahan Tabatabaie. 2022. Spatio-Temporal Graph Attention Embedding for Joint Crowd Flow and Transition Predictions: A Wi-Fi-based Mobility Case Study. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 4, Article 187 (dec 2022), 24 pages. https://doi.org/10.1145/3495003

[45] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials* 21, 3 (2019), 2224–2287.

[46] Lingyan Zhang, Shaohua Wu, Tingting Zhang, and Qinyu Zhang. 2023. Learning to Locate: Adaptive Fingerprint-Based Localization with Few-Shot Relation Learning in Dynamic Indoor Environments. *IEEE Transactions on Wireless Communications* (2023), 1–1. https://doi.org/10.1109/TWC.2022.3232858

[47] Mingxin Zhang, Zipei Fan, Ryosuke Shibasaki, and Xuan Song. 2023. Domain Adversarial Graph Convolutional Network Based on RSSI and Crowdsensing for Indoor Localization. *IEEE Internet of Things Journal* 10, 15 (2023), 13662–13672. https://doi.org/10.1109/JIOT.2023.3262740

[48] Xing Zhang and Wei Sun. 2021. Design of Indoor Localization Algorithm Based on WiFi Fingerprints. In *2021 China Automation Congress (CAC)*. 6209–6214. https://doi.org/10.1109/CAC53003.2021.9728283

[49] Shiyu Zheng, Ziheng Zhou, Qi Zhang, Shanshan Zhang, Ao Peng, Lingxiang Zheng, Huiru Zheng, and Haiying Wang. 2023. A Factor Graph Based Indoor Localization Approach for Healthcare. In *2023 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 3257–3264. https://doi.org/10.1109/BIBM58861.2023.10385673

[50] Xiaoqiang Zhu, Wenyu Qu, Tie Qiu, Laiping Zhao, Mohammed Atiquzzaman, and Dapeng Oliver Wu. 2020. Indoor intelligent fingerprint-based localization: Principles, approaches and challenges. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2634–2657.

[51] Weipeng Zhuo, Ka Ho Chiu, Jierun Chen, Jiajie Tan, Edmund Sumpena, S. H. Gary Chan, Sangtae Ha, and Chul-Ho Lee. 2023. Semi-supervised Learning with Network Embedding on Ambient RF Signals for Geofencing Services. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE.

[52] Weipeng Zhuo, Ka Ho Chiu, Jierun Chen, Ziqi Zhao, S-H Gary Chan, Sangtae Ha, and Chul-Ho Lee. 2023. Fis-one: Floor identification system with one label for crowdsourced rf signals. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 418–428.

[53] Weipeng Zhuo, Ziqi Zhao, Ka Ho Chiu, Shiju Li, Sangtae Ha, Chul-Ho Lee, and S-H Gary Chan. 2022. GRAFICS: Graph Embedding-based Floor Identification Using Crowdsourced RF Signals. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1051–1061.