Enabling Seamless LoRa-to-Mobile Bridging via Wi-Fi Spectrum Scanning

Jiaqi Zhou*, Fengxu Yang*, Yihui Yan†, Zhice Yang*‡
*School of Information Science and Technology, ShanghaiTech University, China
†College of Information Engineering, Shanghai Maritime University, China

***Abstract*—LoRa is a popular low-power wireless technology in the Internet of Things (IoT), but direct interaction with consumer devices is hindered by protocol incompatibility. Existing cross-technology communication (CTC) solutions for Wi-Fi receivers often require additional gateways or transmitters. We propose a LoRa-to-Wi-Fi CTC technique that enables smartphones to directly receive and decode LoRa transmissions using only their native hardware capabilities. Our key innovation is utilizing the spectrum scanning functionality in commercial mobile Wi-Fi chipsets for LoRa packet decoding. A signal processing pipeline combining frequency-domain analysis and lightweight neural-network-based symbol classification reliably recovers LoRa packets under commodity hardware constraints. Our prototype, implemented on a Google Pixel 5, demonstrates robust one-to-one LoRa-to-Wi-Fi communication with commercial LoRa tags, enabling smartphones to act as ad hoc LoRa listeners and simplifying infrastructure-free IoT data collection.**

I. Introduction

The Internet of Things (IoT) increasingly relies on various wireless technologies to connect heterogeneous distributed devices. LoRa is one of the most popular low-power standards, widely used for applications such as environmental monitoring, smart agriculture, and asset tracking. Meanwhile, Wi-Fi remains the dominant wireless communication protocol in consumer devices, especially smartphones, which are ubiquitous and equipped with powerful radios and computing resources. Integrating LoRa sensing capabilities with commodity Wi-Fi devices, particularly smartphones, promises to bridge low-power sensing with mainstream communication infrastructure, enabling seamless IoT data collection and interaction.

Previous research has explored cross-technology communication (CTC) approaches to bridge the gap between heterogeneous wireless protocols. Some methods exploit customized Wi-Fi symbol patterns to adapt to LoRa reception, enabling effective Wi-Fi-to-LoRa communication [1]–[3]. However, the reverse direction, LoRa-to-Wi-Fi, remains less convenient. For example, XFi [4], L2X [5], and LoFi [6] propose transmission schemes that allow LoRa devices to convey information to Wi-Fi receivers by leveraging the interference patterns that LoRa packets introduce into Wi-Fi signals. While effective in principle, these CTC solutions often require the cooperation of auxiliary transmitters, which increases deployment cost and complexity. Furthermore, they typically lack support for flexible, mobile, or infrastructure-free deployments, which are increasingly essential in dynamic IoT environments.

‡Corresponding author.

Our goal is to develop a generic and flexible CTC solution that can run directly on end devices. The key enabling technique of our approach is leveraging Wi-Fi spectrum scanning, a capability that is widely supported in Wi-Fi chipsets but rarely discussed in the context of mobile devices.

Modern Wi-Fi chipsets and protocols employ orthogonal frequency-division multiplexing (OFDM), which relies heavily on fast Fourier transforms (FFT). By performing FFT on received radio signals, the device can obtain the spectral profile of the channel. Spectrum scanning allows a Wi-Fi device to passively monitor the radio frequency spectrum, enabling it to perceive frequency-domain modulated signals such as LoRa, and thereby achieve CTC.

A major challenge here is that, although Wi-Fi chipsets can perform FFT computations very fast (at least once for every OFDM symbol), extracting spectrum scanning results from the chip suffers from severe data bandwidth limitations and temporal jitter. In addition, the frequency-domain resolution of Wi-Fi spectrum scanning is inherently constrained by the number of OFDM subcarriers, typically 512. These constraints significantly limit the achievable temporal and spectral resolution, making direct LoRa demodulation from spectrum scans highly challenging.

To overcome them, we propose an end-to-end demodulation solution consisting of two key components:

- Neural-network-based LoRa demodulation — We design a lightweight neural network model that extracts LoRa-specific features from the FFT spectra, enabling robust decoding even under low temporal and spectral resolution (§V).
- Symbol-aware digital redundancy — Through systematic analysis, we identify bits in LoRa symbols that are most prone to decoding errors. By reverse-engineering the LoRa protocol, we introduce redundant coding at the LoRa tag side, selectively avoiding or reinforcing these weak bits to improve end-to-end reliability (§VI).

We evaluate our approach using Semtech SX1280 LoRa tags operating in the 2.4 GHz band as transmitters, with a Google Pixel 5 smartphone serving as the receiver. Our experimental results demonstrate that our method can seamlessly achieve 300–480 bps CTC from LoRa to Wi-Fi without any assistance from existing infrastructure.

Our method can facilitate new ways for everyday consumers to interact with IoT devices. For example, in a smart home where water, gas or electricity meters upload usage data to

the suppliers over LoRa, our method can enable a resident's smartphone to directly collect usage data from the smart meters with no dependency on additional hardware and only a small software update on the phone (and a lightweight encoder at the LoRa tag). The collected data can then be used for further personalized analysis on the smartphone.

In summary, this paper makes the following contributions:

- We measure the performance of Wi-Fi spectrum scanning on commodity mobile devices.
- We design a neural network for LoRa demodulation to overcome limitations in temporal and spectral resolution.
- We propose an additional encoding layer at the LoRa tag to deliberately improve decoding robustness.
- We conduct thorough feasibility evaluations of our method on commercial smartphones and LoRa tags.

II. LoRa Preliminaries

LoRa employs Chirp Spread Spectrum (CSS) modulation, a technique that encodes information by linearly varying the instantaneous frequency of a carrier signal over a specified bandwidth (BW). This kind of modulation generates chirp signals whose frequency either increases (upchirp) or decreases (downchirp) linearly within the defined frequency range.

Each chirp symbol encodes a number of bits determined by the spreading factor (SF). For LoRa operating in the 2.4 GHz band, SF can be set between 5 and 12, with bandwidth options of 203 kHz, 406 kHz, 812 kHz, and 1625 kHz. Specifically, a chirp symbol encodes SF bits, allowing it to represent one of 2^{SF} possible values. These bits are mapped to a specific starting frequency within the bandwidth, from which the instantaneous frequency increases (or decreases) in steps of $\frac{\mathrm{BW}}{2^{\mathrm{SF}}}$, wrapping around to the other end upon reaching the boundary of the bandwidth, and finally arriving back at the starting frequency. The symbol duration is given by $T_s = \frac{2^{\mathrm{SF}}}{\mathrm{BW}}$.

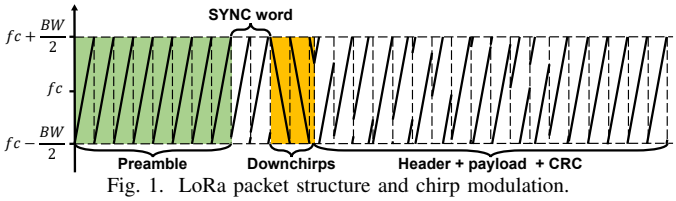

Fig. 1. LoRa packet structure and chirp modulation.

A LoRa packet, as illustrated in Fig. 1, consists of preamble, synchronization word (SYNC word), downchirps, and data section. The preamble comprises a configurable number of upchirps, typically ranging from 8 to 61440, which aims to awake receivers. Following the preamble, the SYNC word, consisting of two upchirps, provides network identification and frame synchronization; its values are set by users or operators (*e.g.,* 0x01 and 0x02 for private networks, or 0x03 and 0x04 for LoRaWAN). This is followed by 2.25 downchirps, used for fine timing adjustment and carrier frequency offset estimation. The data section includes a header (which may be explicit or implicit), the payload, and an optional CRC for error detection. The header can specify payload length, addressing, and control flags. Both payload length and CRC usage are configurable to accommodate different communication requirements.

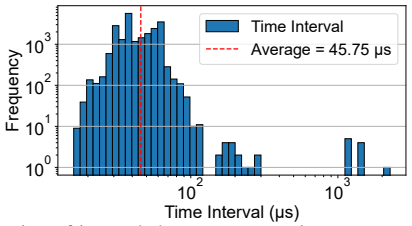

Fig. 2. Distribution of intervals between successive spectrum scan messages, collected within a 1-second window.

III. Properties of Wi-Fi Spectrum Scan

Modern Wi-Fi chipsets widely support spectrum scanning, which enables devices to measure the power distribution across the radio spectrum by performing FFT on received signals. This capability is primarily intended for wireless environment awareness rather than direct data reception. For example, in some regions the 5 GHz ISM band requires devices to support Dynamic Frequency Selection (DFS) to monitor radar signals. In addition, spectrum information is useful for radio resource management, diagnostics, and channel selection.

Several mainstream WLAN NICs have features to report FFT data from the baseband under software controlled conditions, *i.e.,* spectrum scan features, that are exposed through their drivers. Atheros' open-source wireless drivers `ath9k` and `ath10k` expose the spectrum scan interfaces of many Atheros PCIe WLAN NICs via Linux kernel's `debugfs` interface. Meanwhile, on commodity Qualcomm SoC smartphones, Qualcomm's `qcacld` driver makes it possible [7] to have the WLAN NIC perform spectrum scans by sending user space `nl80211` commands, and to receive spectrum scan results via `Generic Netlink`.

On a Qualcomm SoC smartphone, each reported `Generic Netlink` message mainly contains two useful components: (1) a timestamp field in microseconds, and (2) the corresponding FFT power array, with each value represented as an 8-bit integer. For convenience, we refer to a single FFT array at a specific timestamp as an *FFT sample*.

The maximum possible number of FFT bins in a sample is determined by the OFDM subcarrier count (*e.g.,* 256/512 bins for 20/40 MHz channels). This directly fixes the frequency resolution, as the spacing between adjacent bins equals the subcarrier spacing (*e.g.,* 40 MHz / 512 $\approx$ 78 kHz).

Once spectrum scanning starts, the messages are reported to the host in a best-effort manner by the hardware, and in a limited and often irregular pace. Fig. 2 presents the distribution of inter-message intervals measured on a Google Pixel 5 over a 1-second period. Both the time and frequency (count) axes are plotted on a logarithmic scale. The majority of intervals concentrate between 20 μs and 100 μs. A small number of larger intervals occasionally appear as outliers; the causes of these irregularities remain unclear to us.

IV. Sensing LoRa via Wi-Fi Spectrum Scan: A First Look

Wi-Fi spectrum scan (§III) can provide a sequence of FFT samples of 2.4 GHz radio signals. We intend to repurpose this capability for LoRa reception. Although FFT-based spectrum

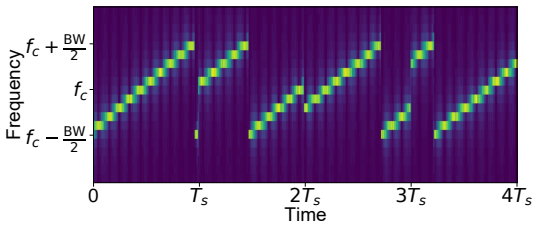

Fig. 3. Spectrogram of four LoRa symbols drawn based on FFT samples.

scans lose the I/Q phase information, LoRa employs CSS modulation (§II), where the instantaneous frequency sweeps linearly over time. As a result, even with power-only snapshots, the movement of LoRa energy peaks across FFT bins can still be observed, revealing the underlying chirp pattern and making demodulation conceptually feasible.

To examine this feasibility, we conduct a preliminary experiment (full testbed setup is described in §VII). A LoRa tag transmits at 2.403 GHz (overlapping Wi-Fi Channel 1) with 812 kHz bandwidth and SF 12, while a smartphone performs 40 MHz spectrum scanning with 512 FFT bins (78 kHz/bin) at roughly 50 μs intervals. Fig. 3 shows a zoomed-in time–frequency spectrogram of the captured scans. The LoRa chirps appear as diagonal ridges sweeping across FFT bins, confirming that Wi-Fi spectrum scanning can sense LoRa's frequency evolution.

A. Practical Constraints

However, several inherent constraints make reliable LoRa packet decoding from the above spectrograms difficult:

Limited Frequency Resolution: One common LoRa demodulation method is to track the instantaneous frequency ramp of each chirp. However, the 78 kHz frequency resolution offered by spectrum scan is orders of magnitude coarser than the fine-grained shifts within LoRa symbols. In the most distinguishable case, SF 5@1625 kHz BW, each symbol spans 2^5=32 discrete frequency steps across the band, resulting in approximately 50.78 kHz per step. With 78 kHz per FFT bin, about 1.5 adjacent chirp states collapse into the same bin, making it impossible to resolve the true instantaneous frequency. This further leads to aliasing of the chirp trajectory and ambiguous symbol boundaries.

Low Temporal Resolution and Large Jitter: LoRa symbols span relatively long durations, which allows symbol discrimination through holistic time–frequency patterns rather than precise instantaneous frequency estimates alone. However, our measurements (Fig. 2) show that the FFT sampling intervals are neither fine-grained nor stable. Even under ideal conditions, a 50 μs sampling interval would provide only about 100 spectrum snapshots over a 5 ms LoRa symbol (*e.g.,* SF 12@812 kHz BW). In practice, large jitter and irregular spacing in the snapshot stream further distort the observed chirp slope, severely degrading both timing synchronization and symbol correlation.

B. Transmitter and Receiver Design Overview

To overcome the above challenges, As shown in Fig. 4, we adopt a two-fold strategy. First, we jointly exploit the time–frequency evolution of LoRa chirps, using a lightweight neural network to map spectrogram segments directly to symbol decisions. The neural network is particularly suitable here because the coarse and jittery spectrograms exhibit complex, non-linear patterns that are difficult for rule-based or correlation methods (§V). Second, we introduce an additional payload-level encoding scheme to mitigate symbol ambiguity, ensuring that potentially confusing symbols are temporally separated, which improves robustness under low-resolution spectrum scanning (§VI).

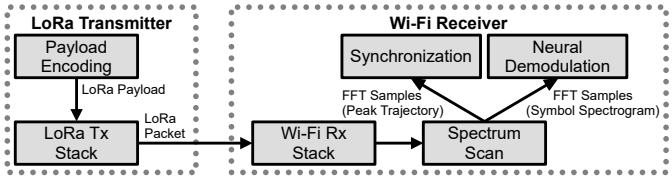

Fig. 4. Workflow of LoRa-to-Wi-Fi communication.

V. Receiver Design

This section describes the receiver at the smartphone.

A. Lightweight Synchronization

LoRa reception requires accurate time and frame synchronization, as the receiver must know where each chirp starts and ends to correctly interpret each symbol. A LoRa packet begins with a preamble of 8 upchirps, two SYNC word symbols, and 2.25 downchirps. In our configuration, the SYNC word symbols are close to 0, so the preamble can be effectively regarded as 10 upchirps followed by 2.25 downchirps, with a total duration of $12.25T_s$, where T_s is the symbol duration.

Instead of performing costly correlation with reference chirps directly on the time–frequency data for synchronization, we adopt a two-step lightweight approach.

First, for each FFT sample with timestamp t_i, we identify the index of the bin with the maximum magnitude, which is then mapped to its center frequency, denoted as f_i. The resulting sequence $\{f_i\}$ forms a coarse time–frequency trajectory of the signal, which serves as the input to subsequent tasks.

Next, we exploit the monotonic frequency evolution of LoRa chirps to detect the preamble in the $\{f_i\}$ sequence. Specifically, We maintain a sliding window of length $12.25T_s$, denoted as $[t_w, t_w + 12.25T_s)$, over the peak-frequency sequence $\{f_i\}$. Within each window, we check whether the observed $\{f_i\}$ exhibits the characteristic preamble pattern: a long run of periodically increasing frequencies corresponding to 10 upchirps, followed by a shorter run of decreasing frequencies corresponding to 2.25 downchirps. To formally validate the existence of such frequency trends, we define the following two conditions:

- **Upchirp detection:** In the first $10T_s$ of the window ($t_i \in [t_w, t_w + 10T_s)$), the number of samples with $f_i > f_{i-1}$ must be at least 100, and more than twice the number with $f_i < f_{i-1}$.
- **Downchirp detection:** In the last $2.25T_s$ of the window ($t_i \in [t_w + 10T_s, t_w + 12.25T_s)$), the number of samples with $f_i < f_{i-1}$ must be at least 20, and more than twice the number with $f_i > f_{i-1}$.

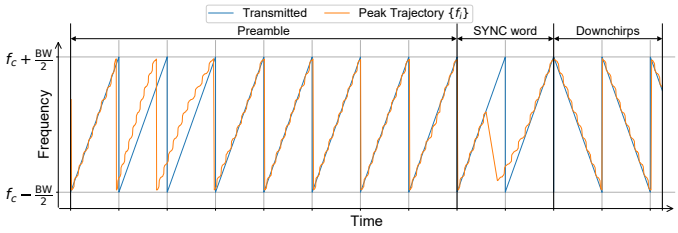

Fig. 5. Example peak-frequency trajectory (a LoRa preamble segment).

Once a valid preamble is detected, we further refine its position to obtain precise symbol boundaries. We slide a short window around the detected preamble and compute a local alignment score that quantifies the consistency of the upchirp or downchirp slope. The timestamp that maximizes this score is selected as the most probable start of a chirp, which then serves as the reference for partitioning the subsequent spectrum samples into T_s-length segments for symbol demodulation.

The above method can tolerate considerable frequency and timing errors. For example, in Fig. 5, the second and third upchirps in the preamble exhibit noticeable time–frequency deviations, and the first SYNC word symbol even contains a large data gap due to message interval outliers. Despite these imperfections, our method is still able to correctly infer the symbol boundaries with reasonable accuracy.

B. Neural Demodulation

While synchronization is relatively tolerant to occasional errors by leveraging multiple preamble symbols, symbol decoding is more challenging, as it requires accurate interpretation of the frequency evolution within each individual symbol. Therefore, instead of using the peak-frequency trajectory, we utilize the full time–frequency spectrogram data to achieve more precise decoding. Based on the synchronization results, we divide the spectrogram into symbol-aligned chunks according to the estimated symbol boundaries for joint time–frequency decoding.

Our method is inspired by successful convolutional neural networks (CNNs) in image recognition tasks, where CNNs extract spatial features such as edges and textures. Similarly, the time–frequency spectrogram of LoRa signals can be viewed as a 2D image with patterns corresponding to chirp modulation. By applying convolutional filters, our model effectively captures the characteristic frequency sweeps and temporal correlations of LoRa symbols, enabling more accurate demodulation under noisy and irregular sampling conditions.

The structure of the CNN model is summarized in Table I, where N denotes the batch size. The demodulation of each symbol proceeds as follows:

1) **Normalization:** For each FFT sample within a symbol duration, we consider the bins within twice the LoRa BW around the transmitter's center frequency. The magnitudes of these bins are normalized by the maximum magnitude among them.
2) **Interpolation:** The normalized samples over the symbol duration are linearly interpolated along the time axis to obtain 128 uniformly spaced samples.

TABLE I
LAYERS OF THE CNN MODEL

Layer Type	Output Shape
Input	$N \times 1 \times 128 \times 20$
Convolution (3×3 kernel, $1 \to 32$ channels)	$N \times 32 \times 128 \times 20$
Batch Normalization (32 channels)	$N \times 32 \times 128 \times 20$
ReLU	$N \times 32 \times 128 \times 20$
Max Pooling (2×1 kernel)	$N \times 32 \times 64 \times 20$
Convolution (3×3 kernel, $32 \to 64$ channels)	$N \times 64 \times 64 \times 20$
Batch Normalization (64 channels)	$N \times 64 \times 64 \times 20$
ReLU	$N \times 64 \times 64 \times 20$
Max Pooling (2×2 kernel)	$N \times 64 \times 32 \times 10$
Convolution (3×3 kernel, $64 \to 64$ channels)	$N \times 64 \times 32 \times 10$
Batch Normalization (64 channels)	$N \times 64 \times 32 \times 10$
ReLU	$N \times 64 \times 32 \times 10$
Max Pooling (2×2 kernel)	$N \times 64 \times 16 \times 5$
Flattening	$N \times 5120$
Dropout (50% rate at training time only)	$N \times 5120$
Linear ($5120 \to 2^k$ features)	$N \times 2^k$

3) **Classification:** The interpolated samples are arranged into a $1 \times 128 \times 20$ array and fed in batches of N symbols to the CNN, which classifies each symbol into one of 2^k classes, where the class label is a integer in the range $[0, 2^k)$ and corresponds to the top k bits of the symbol value[1].

C. Most Significant Bits (MSBs) Attention Mechanism

We deliberately design the CNN to classify only the top k bits of each symbol rather than the full symbol space to improve the robustness of demodulation. This is because changes in the bottom bits, *i.e.,* least significant bits (LSBs), of a LoRa symbol produce only very small frequency shifts, corresponding to slight displacements in the time–frequency spectrogram. As a result, symbols that differ only in their LSBs are visually and numerically very similar, making them highly susceptible to noise and misclassification.

By focusing on the most significant k bits (MSBs), the CNN classification can achieve reasonable classification results even when the lower bits are ambiguous. In practice, we fix $k = 4$ to balance robustness and information rate. However, commercial LoRa transmitters do not rely solely on the top 4 bits to encode the payload. Therefore, in the next section, we introduce an additional data encoding layer that maps the original data to symbols in a way that only utilizes the MSBs, mitigating the ambiguity caused by the LSBs.

VI. TRANSMITTER DESIGN

Our software payload encoder is responsible for converting the content to be transmitted into the proper LoRa packet payload. The encoder takes as input a sequence of 4-bit values, which are intended to appear as the most significant bits (MSBs) of the transmitted LoRa symbols, and produces a standard LoRa payload.

[1]Parameter k of CNN defines the effective decoding resolution of the model. If the actual LoRa symbol space is smaller than 2^k, multiple classes may map to the same symbol. If the symbol space is larger than 2^k, the model only decodes the top k bits of the symbol value. For example, when $k = 6$, the CNN outputs 64 classes: symbol space 0–63 is represented exactly, while symbol 200 (11001000_2) in a symbol space of 0–255 would be classified according to its top 6 bits 110010_2 as class 50.

However, LoRa symbols are not directly derived from the payload bytes; instead, they are generated through a multi-stage processing pipeline as below: [8]

1) **Whitening:** Each payload byte is XORed with a pseudo random value determined solely by its byte position, then split into two nibbles (4-bit/half-byte data).
2) **Header and CRC Insertion:** A 5-nibble header and an optional CRC checksum are inserted at the start and the end respectively. We disable the hardware CRC since it cannot be correctly verified in our setup, and instead append CRC bits in the payload for upper layer error detection.
3) **Forward Error Correction (FEC):** LoRa applies a Hamming-like block code to each nibble. For a coding rate of $4/(4+P)$, $P \in \{1,2,3,4\}$ parity bits are added to a nibble to form a $(4+P)$ bit codeword.
4) **Interleaving:** Groups of SF (or SF $-$ 2) consecutive FEC codewords are rearranged by a diagonal block interleaver (Fig. 6), producing $4+P$ bit vectors of SF bits each. This step disperses burst errors and improves FEC performance.
5) **Gray Demapping:** Finally, each interleaved bit vector I is mapped to a LoRa symbol s using inverse Gray coding, where $s \oplus (s \gg 1) = I$. This step is applied to minimize the number of bit errors when a symbol is misclassified.

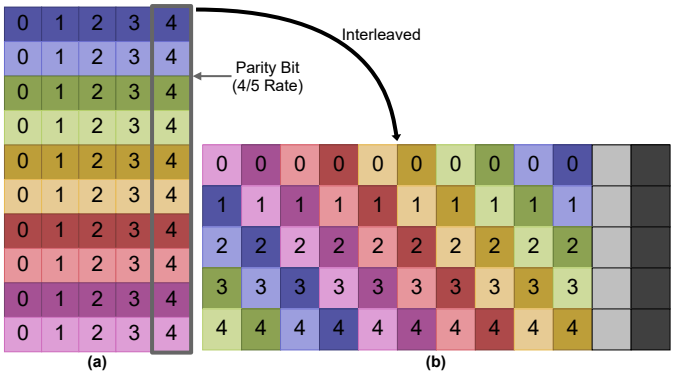

Fig. 6. Illustration of interleaving with SF = 12 and CR = 4/5. (a) Codewords after FEC. (b) Bit vectors produced by interleaving. Different colors indicate bits belonging to different 5-bit codewords, and the number in each square denotes the bit position within the codeword.

A. Payload Encoder Pipeline

Our payload encoder inverts the above stages to synthesize valid payload bytes whose transmitted symbols have the desired MSBs. Without loss of generality, we illustrate below using SF = 12 and CR = 4/5.

1) Gray Mapping: We begin with the desired MSBs of each 12-bit LoRa symbol. For example, to transmit the sequence 0x1, 0x2, 0x3, 0x4, 0x5, we fix the upper 4 bits of each symbol to 0001_2, 0010_2, …, 0101_2 and temporarily set the lower 8 bits to "don't-care" (*e.g.,* $0011********_2$). We apply Gray mapping to obtain a 12-bit bit vector for each symbol.

2) Inverse Interleaving: We then concatenate every 5 12-bit vectors into an interleaved bit block (*e.g.,* 5 symbols $\rightarrow$ 60 bits). Using the LoRa interleaver matrix (Fig. 6), we perform the inverse permutation to redistribute these bits into their original 5-bit FEC codewords. Note that when Low Data Rate Optimization (LDRO) is enabled, the last two bits in each bit vector serve a different purpose, which can be ignored during this step as they are "don't-care" bits.

3) Filling Remaining Bits: Each 5-bit codeword contains one or more "don't care" bits after the last step. We want to limit the possible assignments of a symbol's 8 "don't-care" bits in order to increase the distinction between symbols with different MSBs, while the FEC included in each codeword also imposes limitations on the "don't-care" bits. For CR = 4/5, each 5-bit codeword is systematic: the first 4 bits are data, and the last bit is the XOR parity of those 4 bits. In this step, we assign values to the "don't care" bits so that XORing the 5 bits of any codeword yields 0, while in each symbol the 8 "don't care" bits begin with either 01111_2 or 10000_2. Finally, concatenating the first 4 bits of each codeword yields the desired bitstream for the next step.

CTC Header 8 symbols from the first interleaver block	CTC Payload 5 symbols from each other interleaver block	CTC Padding 5 symbols from the last interleaver block

Fig. 7. Symbols in the data section of a LoRa packet in the view of CTC.

4) Handling Header and Padding: LoRa arranges codewords into interleaver blocks that generate corresponding symbols. The first interleaver block is special: it has an effective coding rate of 4/8 and contains 10 codewords, with the first 5 representing the LoRa PHY header and the next 5 corresponding to the first 5 payload nibbles. This block produces 8 symbols, which we collectively refer to as the CTC header. Due to complex parity and CRC constraints on the header codewords, the transmitter's control over the MSBs of these symbols is limited, so only limited metadata embedding is possible in this region.

Subsequent interleaver blocks each handles 10 payload nibbles and produces 5 symbols, whose MSBs form the main CTC payload area where we embed desired symbol sequences by applying inverse LoRa PHY operations. Since payloads are byte-aligned, the total number of nibbles must be even, and at least 1 padding nibble have to be added. This results in an additional block producing 5 extra symbols which we refer to as the CTC padding, and generally aren't used for primary CTC data embedding.

5) Inverse Whitening: Whitening is implemented as a bit-wise XOR with a pseudorandom sequence. To invert whitening, we precalculate the identical pseudorandom sequence for the payload length and XOR it bitwise with the bitstream from the previous step. After this operation, we obtain the desired payload bits compatible for CTC transmission.

VII. Implementation

Fig. 8 shows the prototype of the CTC system. It also serves as the testbed for evaluation.

Receiver. We consider two deployment settings: (1) an offline setup utilizing a GPU for decoding, and (2) an on-device setup running entirely on a Google Pixel 5 smartphone to demonstrate real-time capabilities. The only difference between these settings lies in the decoding stage, which uses GPUs of varying computational power. In the offline setup, spectrum scanning is performed on the smartphone, but FFT

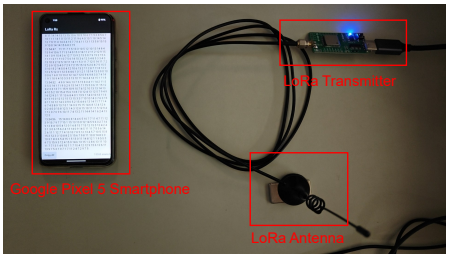

Fig. 8. Prototype of the system.

samples are transferred to a laptop PC for decoding. In the on-device setup, all operations are executed on the smartphone. We configure the smartphone's Wi-Fi to work with a 20 MHz bandwidth and operate at the 2412 MHz band, which overlaps with the LoRa operational band mentioned below.

As long as the WLAN NIC has the spectrum scan feature, the receiver only needs a software update on the smartphone, and on a Google Pixel 5 it's a simple app with root access.

CNN Model Training. We use the receiver to collect FFT samples of LoRa packets, which are subsequently exported for offline decoding and training. Two types of FFT sample datasets are collected: one with payload encoding applied at the LoRa transmitter, and one without, to evaluate the effectiveness of the encoding strategy (§VI). Each dataset consists of 320 packets for training and 64 packets for validation. In the dataset with transmitter payload encoding, each packet contains 210 CTC payload symbols, while in the other dataset each packet contains 203 LoRa payload symbols. Model training on both datasets uses the same neural network with the Adam optimizer. The number of training epochs is set to 30. Training and offline inference are done on a laptop with an NVIDIA MX550 GPU.

LoRa Transmitter. We employ the Semtech SX1280 LoRa transceiver as the transmitter. The module is connected to a host computer via USB, which supplies the bytes to be transmitted. On the host computer, we implement our payload encoding scheme (§VI), which converts arbitrary raw input data into a reliably decodable byte stream. In practice, the host computer can be replaced by any microcontroller (MCU) to serve as an IoT data source. The transmitter is configured with center frequency f_c = 2403 MHz, SF = 12, BW = 812 kHz, and transmit power = 13 dBm.

VIII. Evaluation

In the evaluation, we aim to answer the following questions:

1) How many of the most significant bits in each symbol should be selected to serve as reliable carriers for data transmission? (§VIII-A)
2) How does the performance vary across different receiver settings, *i.e.,* **offline**, on-device while **charging**, and on-device on **battery** power? (§VIII-B)
3) What is overall system performance when applying our approach, *i.e.,* PRR and throughput? (§VIII-C)
4) How well does our approach generalize across different LoRa center frequencies in the 2.4 GHz band? (§VIII-D)
5) What is the system cost in terms of computational overhead (§VIII-E) and energy consumption (§VIII-F)?

Metrics. We evaluate system performance using three metrics: (1) **symbol error rate** (SER), defined as the ratio of incorrectly received symbols; (2) **packet reception rate** (PRR), defined as the ratio of packets with all symbols correctly received; and (3) **throughput**, which measures the number of effective bits transmitted per second.

Environment. We conduct experiments in a 10 m $\times$ 7.5 m office room, with the transmitter's antenna placed 1 m from the nearest wall and 2 m from the nearest corner. Regular ambient traffic is present during the experiments, produced by 3–6 people using various personal computers, wearables and smartphones inside the room, as well as by at least 10 access points and sensors both inside and outside the room.

A. Top k-bit Selection

To determine how many top-ranked bits within a symbol should be retained for optimal decoding, we train and evaluate 10 CNN models, each configured with $k = 1, 2, 3, 4, \ldots, 10$ respectively, on the dataset without transmitter payload encoding. The accuracy of each model is shown in Fig. 9, while the macro-average precision and recall are within $\pm 0.1\%$ of the accuracy for all models. The accuracy is $\geq 90\%$ for $k \leq 4$, and drops by over 7% for each increase in k when $k \geq 4$.

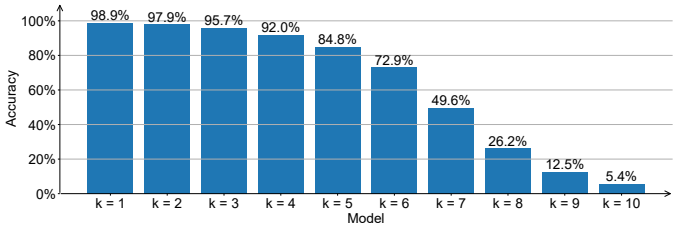

Fig. 9. Each model's accuracy of demodulating top k bits of a symbol.

We then consider the dataset with transmitter payload encoding. On this dataset, the previously trained model with $k = 4$ achieves 98.1% validation accuracy, which is notably higher than the 92.0% mentioned above. We train another CNN model with $k = 4$ on this dataset, and it achieves slightly higher validation accuracy of 98.5%, and 99.3% accuracy on the training set. This model's confusion matrix on the validation set is shown in Fig. 10, and it will be the only CNN model used in the following experiments.

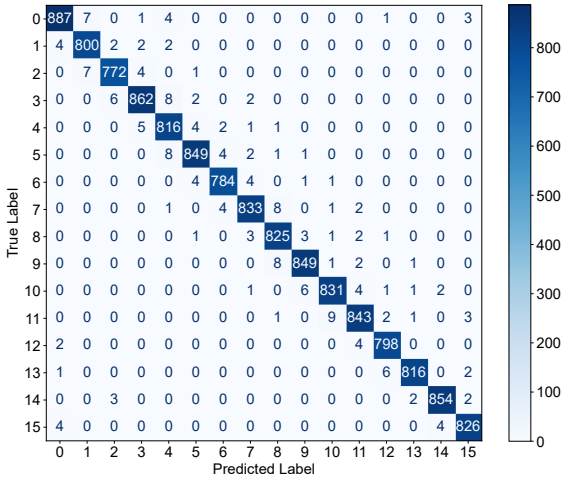

Fig. 10. Confusion matrix on dataset with transmitter payload encoding.

B. Symbol Error Rate

For SER measurement, we transmit LoRa packets each containing 210 CTC payload symbols (or 233 LoRa payload symbols). Each packet lasts about 1.186 seconds, and we transmit one packet every 4.41 seconds, so the transmitter's duty cycle is about 26.9%.

We measure the SER with the 3 different ways of running the receiver as well as 3 different received signals strengths achieved by adjusting the transmitter's power level (between 13 dBm and 3 dBm) and the distance between the transmitter and the receiver (between 0.5 m and 4 m). In each of these 9 cases, we transmit 1600 packets, calculate the SER in the 210 CTC payload symbols of each individual packet, and then compute statistics like mean and CDF across the 1600 packets. If a packet isn't detected by the receiver at all, we consider all its symbols to be incorrectly demodulated. The mean SER values are shown in Fig. 11, and the CDFs of the SER in 1600 packets are shown in Fig. 12.

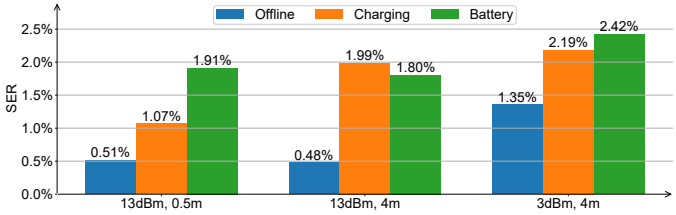

Fig. 11. Mean SER in 1600 packets.

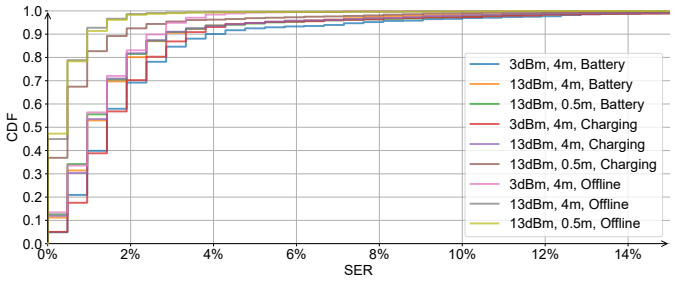

Fig. 12. CDFs of the SER in 1600 packets.

The SER when processing the samples in real time is higher than when processing the samples **offline**. The exact cause of this phenomenon is unclear, but it's possible that the real-time computations lead to more timestamp errors in FFT samples. At 0.5 m distance, the SER is also higher when the phone is on **battery** than when the phone is **charging**, which can be caused by energy saving mechanisms in either the hardware or the operating system. The SER in the **offline** case can be even lower than the CNN model's SER on the training set, which may be the result of a change in the phone's hardware conditions or environmental factors that we do not control.

It is also noticeable that when the distance increases from 0.5 m to 4 m while keeping the transmitter's power level at 13 dBm, the mean SER increases in the **charging** case, but slightly decreases in the other two cases. Meanwhile, when reducing the transmitter's power level from 13 dBm to 3 dBm while keeping the distance at 4 m, the increase in mean SER is much smaller in the **charging** case than in the other two cases. A possible explanation for this phenomenon is that the combined electromagnetic interference (EMI) of real-time computation and charging is stronger and starts affecting the FFT samples at a higher received LoRa signal strength.

C. Packet Reception Rate and Throughput

Considering that each CTC payload symbol consists of 4 bits, a suitable error correction code for this scheme is the Reed-Solomon code, which supports symbols of various bit lengths. For 4-bit symbols, the maximum possible block size of Reed-Solomon code is $2^4 - 1 = 15$ symbols. We choose three different Reed-Solomon coding schemes: $\mathrm{RS}(15, 13)$, $\mathrm{RS}(15, 11)$, and $\mathrm{RS}(15, 9)$, where the number of data symbols per block are 13, 11, and 9, respectively, and the remaining symbols serve as parity.

Since incorrect demodulation often occurs in bursts, we apply interleaving and distribute 14 Reed-Solomon blocks across the 210 CTC payload symbols in each packet. This helps spread burst errors across multiple blocks and make error correction more likely to succeed. Consequently, a packet contains $14 \times 13 = 182$, $14 \times 11 = 154$, or $14 \times 9 = 126$ data symbols respectively when using each of the three Reed-Solomon schemes. For each scheme, we transmit 1200 such packets and apply a Reed-Solomon decoder of the same configuration at the receiver to correct symbol errors. If a block contains errors that cannot be corrected, the decoder falls back to the original data symbols of that block, effectively discarding the parity symbols.

We evaluate PRR only under the **charging** receiver mode, while keeping all other configurations consistent with those used for SER measurement. The resulting PRR values under different Reed-Solomon schemes and varying combinations of distance and transmitter power are shown in Fig. 13.

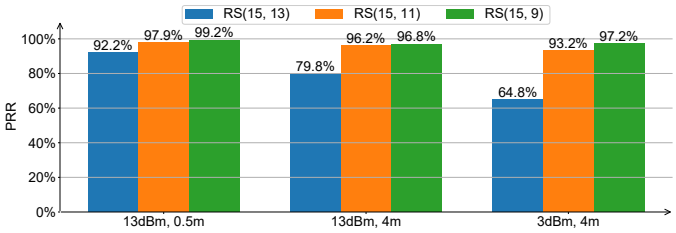

Fig. 13. PRR with different error correction, distances and power levels.

$\mathrm{RS}(15, 11)$ can achieve the highest throughput in the two cases with 4 m distance. If we increase the transmitter duty cycle from 26.9% to 62.1% and repeat the experiment with $\mathrm{RS}(15, 11)$, 3 dBm and 4 m, the PRR drops slightly to 92.7%. Each packet contains $154 \times 4 = 616$ data bits in this case, resulting in a throughput of roughly 300 bps. The throughput can reach 480 bps in the ideal case (100% transmitter duty cycle), which may require optimizations (*e.g.,* hardware CNN inference acceleration) that reduce the receiver's CPU usage.

D. Generalization Across Different Frequencies

All previous experiments are done with the transmitter's center frequency f_c set to 2403 MHz. We then measure the SER with various different f_c values in the range 2403–2488 MHz, while still using the same CNN model and parameters for demodulation. We use 3 dBm transmitter power, keep 4 m distance between the transmitter and the receiver, and transmit

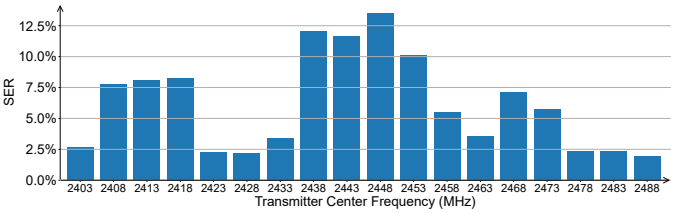

Fig. 14. SER with different transmitter center frequencies.

400 packets for each f_c. We set the phone's Wi-Fi center frequency to $f_c + 9$ MHz if $f_c \leq 2463$ MHz, or $f_c - 16$ MHz otherwise. The measured SER values are shown in Fig. 14.

The SER can be even lower at some center frequencies than at 2403 MHz, so the receiver is able to generalize across different center frequencies. However, at several other center frequencies the SER is several times higher, and there are dozens of packets that either have almost all CTC payload incorrectly demodulated or aren't detected at all, which may occur when ambient traffic collides with LoRa preambles. This performance variation may indicate non-uniform behavior of the phone's hardware, or that the CNN model's training dataset doesn't cover enough types of ambient traffic, as it's collected with 2403 MHz transmitter center frequency, which is near the boundary of the 2.4 GHz band.

All these tested f_c values satisfy that $f_c - 2403$ MHz is a multiple of the FFT bin width 78125 Hz. If this condition isn't satisfied, a CNN model trained on a different dataset may be needed for demodulation.

E. Receiver Computation Overhead

We measure the time taken by receiver pipeline to process FFT samples for CTC by running it offline on a 2023 laptop with a 13th Gen Intel Core i7 CPU and an NVIDIA GeForce MX550 GPU. When using ExecuTorch with XNNPACK backend for CNN inference, the receiver takes 353 seconds to process FFT samples of the same 1600 packets as when measuring the SER, and all 16 CPU cores are used during the demodulation step. If we remove the demodulation step from the receiver pipeline, the remaining part takes 118 seconds, which is 33.4% of the total time, so demodulation takes 66.6% of the total time. Steps other than demodulation only use 1 CPU core. The average time taken to process a single packet is 0.221 seconds, or 18.6% of a packet's duration.

If we instead use LibTorch with CUDA acceleration for CNN inference, the demodulation is mostly done on the GPU and the entire pipeline only uses 1 CPU core, but takes 26.1% more time than ExecuTorch to process the same 1600 packets. The additional overhead may come from LibTorch supporting more features, or from data transfer between CPU memory and GPU memory.

F. Receiver Battery Consumption

We measure the battery consumption of the receiver by running the receiver Android app while the phone is on battery and obtaining the system's battery statistics afterwards. We the use difference between the battery drain when running the receiver and the battery drain when not running any app as the receiver's total battery consumption, which can be further divided into two parts:

- Battery consumed by FFT sample processing. This part includes synchronization, neural demodulation, etc., and is mostly equivalent to the part of the app run as a non-root UID specific to the app. This UID's battery consumption can hence be used here.
- Battery consumed by FFT sample collection. This part includes the WLAN NIC producing FFT samples, the WLAN driver forwarding FFT samples to the user space, and the app receiving FFT samples from the driver. The battery consumption isn't straightforward to obtain here as it spans hardware, kernel space and user space, so we simply subtract the battery consumption of the other part from the receiver's total battery consumption.

We then use the product of the battery consumption (in mAh unit) and the battery voltage of 3.87 V as the energy consumption. We measure the battery and energy consumption of both an idle receiver that runs for 30 minutes without receiving any packets, and a busy receiver that receives 400 packets (each containing 210 CTC payload symbols) over a duration of 30 minutes, which are listed in Table II and Table III respectively.

TABLE II
BATTERY/ENERGY CONSUMPTION OF AN IDLE RECEIVER

	Total	Collection	Processing
Total Battery Consumption	196 mAh	186.8 mAh	9.2 mAh
Total Energy Consumption	2730.7 J	2602.5 J	128.2 J
Per Second Energy Consumption	1.52 J	1.45 J	71.2 mJ

TABLE III
BATTERY/ENERGY CONSUMPTION OF A BUSY RECEIVER

	Total	Collection	Processing
Total Battery Consumption	230 mAh	194.5 mAh	35.5 mAh
Total Energy Consumption	3204.4 J	2709.8 J	494.6 J
Per Packet Energy Consumption	8.01 J	6.77 J	1.24 J

Even though FFT sample processing uses a CNN which is quite computation-intensive, FFT sample collection still consumes several times more battery. This indicates that Qualcomm WLAN NIC's spectrum scan feature itself can be quite energy-consuming when reporting results at the shortest possible interval, and makes our method more suitable for interactive or scheduled usage than for passive reception.

IX. Related Work

CTC Methodologies. Methods for cross-technology communication (CTC) can be categorized based on the granularity of their utilized information: packet-level and physical-level. Packet-level CTC encodes data using attributes such as packet size [9], [10], scheduling [11], [12], and energy [13]–[16]. The encoded data is then decoded at the receiver by monitoring signal indicators like Received Signal Strength (RSS) [11]–[15] or Channel State Information (CSI) [16]–[19]. Physical-level CTC achieves higher data rates and efficiency by encoding data within packet payloads. These methods typically require either a transmitter capable of emulating receiver signals by

adjusting payload content [1]–[3], [20]–[23], or a sophisticated receiver capable of directly decoding transmitter payloads [4], [6], [24], [25].

CTC Between LoRa and Narrowband Protocols. Prior research explores CTC between LoRa and narrowband protocols such as Bluetooth, BLE, and Zigbee. Symphony [26] generates continuous single-tone sine waves detectable by LoRa by manipulating BLE/Zigbee payloads. LoRaBee [27] places specific bytes in LoRa payloads for detection by Zigbee via RSS variations. BLE2LoRa [28] selects BLE bits to mimic LoRa modulation. EMU [29] enables non-LoRa modulators (FSK, GFSK, OOK) to emulate LoRa chirps. LigBee [30] identifies LoRa symbols from phase patterns at Zigbee receivers. ZigRa [31] applies neural networks to map Zigbee payloads into emulated LoRa symbols.

CTC from Wi-Fi to LoRa. These methods primarily focus on Wi-Fi transmitters emulating LoRa signals. Wi-Lo [1], WiRa [2], and WiLo [3] modify Wi-Fi payloads (802.11b, 802.11ax, and 802.11g respectively) to replicate LoRa chirps. XiTuXi [23] generalizes this by employing neural machine translation (NMT), converting Wi-Fi payloads to emulate various IoT signals, including LoRa.

CTC from LoRa to Wi-Fi. Key prior studies in LoRa-to-Wi-Fi communication include XFi [4], LoFi [6], and L2X [5]. XFi [4] leverages packet collisions with Wi-Fi for demodulation, LoFi [6] reconstructs LoRa waveforms via neural networks analyzing collision patterns, and L2X [5] employs synchronized LoRa transmitters for signal superposition, enabling non-LoRa devices to decode the signal. However, these methods require the cooperation of additional transmitters or dedicated hardware.

Comparison of our work. In contrast, our proposed LoRa-to-Wi-Fi approach uniquely facilitates direct communication from LoRa devices to smartphones over Wi-Fi without additional hardware, eliminating the dependency on supplementary transmitters or access points. Furthermore, by directly interfacing with widely available smartphones, our approach enhances usability and flexibility, facilitating rapid deployment and seamless integration into existing Wi-Fi infrastructures.

X. Conclusion

We present a practical LoRa-to-Wi-Fi CTC framework that operates fully on commodity smartphones without extra infrastructure. By enabling spectrum scanning, leveraging neural-network-based demodulation, and applying symbol-aware redundancy, our method achieves 300–480 bps reliable decoding despite low temporal and spectral resolution. This work demonstrates the feasibility of infrastructure-free LoRa-to-Wi-Fi communication and paves the way for real-time, mobile IoT gateways.

References

[1] P. Gawłowicz, A. Zubow, and F. Dressler, "Wi-lo: Emulation of lora using commodity 802.11b wifi devices," in *IEEE ICC*, 2022.

[2] X. Zheng, D. Xia, F. Yu, L. Liu, and H. Ma, "Enabling cross-technology communication from wifi to lora with ieee 802.11ax," *IEEE/ACM TON*, vol. 32, no. 3, pp. 1936–1950, 2024.

[3] D. Gao, H. Wang, S. Wang, W. Wang, Z. Yin, S. Mumtaz, X. Li, V. Frascolla, and A. Nallanathan, "Wilo: Long-range cross-technology communication from wi-fi to lora," *IEEE TCOM*, vol. 73, no. 3, pp. 1677–1691, 2025.

[4] R. Liu, Z. Yin, W. Jiang, and T. He, "Xfi: Cross-technology iot data collection via commodity wifi," in *IEEE ICNP*, 2020.

[5] S. Tong, Y. He, Y. Liu, and J. Wang, "De-spreading over the air: long-range ctc for diverse receivers with lora," in *ACM MobiCom*, 2022.

[6] D. Gao, W. Jiang, R. Liu, W. Wang, Z. Han, and Y. Liu, "Lofi: Physical-layer ctc from lora to wifi with ieee 802.11ax," in *IEEE INFOCOM*, 2025.

[7] J. Zhou, Y. Hang, S. Liao, and Z. Yang, "Enabling radio spectrum scan with smartphones," in *IEEE INFOCOM Workshops*, 2025.

[8] T. Joachim and A. Burg, "Design and implementation of lora physical layer in gnu radio," in *Proceedings of the GNU Radio Conference*, 2024.

[9] K. Chebrolu and A. Dhekne, "Esense: communication through energy sensing," in *ACM MobiCom*, 2009.

[10] Y. Zhang and Q. Li, "Howies: A holistic approach to zigbee assisted wifi energy savings in mobile devices," in *IEEE INFOCOM*, 2013.

[11] S. M. Kim and T. He, "Freebee: Cross-technology communication via free side-channel," in *ACM MobiCom*, 2015.

[12] Z. Yin, W. Jiang, S. M. Kim, and T. He, "C-morse: Cross-technology communication with transparent morse coding," in *IEEE INFOCOM*, 2017.

[13] S. Yin, Q. Li, and O. Gnawali, "Interconnecting wifi devices with ieee 802.15.4 devices without using a gateway," in *DCOSS*, 2015.

[14] X. Guo, Y. He, and X. Zheng, "Wizig: Cross-technology energy communication over a noisy channel," *IEEE/ACM TON*, vol. 28, no. 6, pp. 2449–2460, 2020.

[15] X. Zheng, Y. He, and X. Guo, "Stripcomm: Interference-resilient cross-technology communication in coexisting environments," in *IEEE INFOCOM*, 2018.

[16] Z. Chi, Y. Li, H. Sun, Y. Yao, Z. Lu, and T. Zhu, "B2w2: N-way concurrent communication for iot devices," in *ACM SenSys*, 2016.

[17] X. Guo, Y. He, X. Zheng, L. Yu, and O. Gnawali, "Zigfi: Harnessing channel state information for cross-technology communication," *IEEE/ACM TON*, vol. 28, no. 1, pp. 301–311, 2020.

[18] Z. Chi, Y. Li, Z. Huang, H. Sun, and T. Zhu, "Simultaneous bi-directional communications and data forwarding using a single zigbee data stream," in *IEEE INFOCOM*, 2019.

[19] D. Xia, X. Zheng, L. Liu, C. Wang, and H. Ma, "c-chirp: Towards symmetric cross-technology communication over asymmetric channels," *IEEE/ACM TON*, vol. 29, no. 3, pp. 1169–1182, 2021.

[20] Z. Li and T. He, "Webee: Physical-layer cross-technology communication via emulation," in *ACM MobiCom*, 2017.

[21] Y. He, X. Guo, J. Zhang, and H. Jiang, "Wide: Physical-level ctc via digital emulation," *IEEE/ACM TON*, vol. 29, no. 4, pp. 1567–1579, 2021.

[22] H.-W. Cho and K. G. Shin, "Bluefi: bluetooth over wifi," in *ACM SIGCOMM*, 2021.

[23] S. Liao, Z. An, Q. Pan, X. Zhao, J. Tong, and L. Yang, "Xituxi: Sealing the gaps in cross-technology communication by neural machine translation," in *ACM SenSys*, 2024.

[24] W. Jiang, S. M. Kim, Z. Li, and T. He, "Achieving receiver-side cross-technology communication with cross-decoding," in *ACM MobiCom*, 2018.

[25] X. Guo, Y. He, X. Zheng, Z. Yu, and Y. Liu, "Lego-fi: Transmitter-transparent ctc with cross-demapping," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6665–6676, 2021.

[26] Z. Li and Y. Chen, "Achieving universal low-power wide-area networks on existing wireless devices," in *IEEE ICNP*, 2019.

[27] J. Shi, D. Mu, and M. Sha, "Lorabee: Cross-technology communication from lora to zigbee via payload encoding," in *IEEE ICNP*, 2019.

[28] Z. Li and Y. Chen, "Ble2lora: Cross-technology communication from bluetooth to lora via chirp emulation," in *IEEE SECON*, 2020.

[29] F. Yang, P. Tian, X. Ma, C. A. Boano, Y. Liu, and J. Wei, "Emu: Increasing the performance and applicability of lora through chirp emulation, snipping, and multiplexing," in *ACM/IEEE IPSN*, 2022.

[30] Z. Wang, L. Kong, L. Shangguan, L. He, K. Xu, Y. Cao, H. Yu, Q. Xiang, J. Yu, T. Ma, Z. Song, Z. Liu, and G. Chen, "Ligbee: Symbol-level cross-technology communication from lora to zigbee," in *IEEE INFOCOM*, 2023.

[31] D. Gao, Y. Chen, Y. Liu, and H. Wang, "Seamless physical-layer cross-technology communication from zigbee to lora via neural networks," *IEEE TMC*, vol. 24, no. 11, pp. 11 369–11 385, 2025.