A Low-Density Parity-Check Coding Scheme for LoRa Networking

KANG YANG, University of California Merced, Merced, United States
WAN DU, University of California Merced, Merced, United States

This article presents a novel system, ***LLDPC***,[1] which brings Low-Density Parity-Check (LDPC) codes into Long Range (LoRa) networks to improve Forward Error Correction, a task currently managed by less efficient Hamming codes. Three challenges in achieving this are addressed: First, Chirp Spread Spectrum (CSS) modulation used by LoRa produces only hard demodulation outcomes, whereas LDPC decoding requires Log-Likelihood Ratios (LLR) for each bit. We solve this by developing a CSS-specific LLR extractor. Second, we improve LDPC decoding efficiency by using symbol-level information to fine-tune LLRs of error-prone bits. Finally, to minimize the decoding latency caused by the computationally heavy Soft Belief Propagation (SBP) algorithm typically used in LDPC decoding, we apply graph neural networks to accelerate the process. Our results show that *LLDPC* extends default LoRa's lifetime by 86.7% and reduces SBP algorithm decoding latency by 58.09×.

CCS Concepts: • **Networks** → **Network protocol design**; • **Computing methodologies** → **Neural networks;**

Additional Key Words and Phrases: Wireless Systems, Low-Power Wide-Area Networks, LoRa, Forward Error Correction, Graph Neural Networks

ACM Reference Format:
Kang Yang and Wan Du. 2024. A Low-Density Parity-Check Coding Scheme for LoRa Networking. *ACM Trans. Sensor Netw.* 20, 4, Article 98 (July 2024), 29 pages. https://doi.org/10.1145/3665928

1 INTRODUCTION

The emergence of **Low-Power Wide-Area Networks** (**LPWANs**) has paved the way for connecting billions of affordable **Internet of Things** (**IoT**) devices to facilitate data collection in diverse applications, such as smart industry and precision agriculture [2–5]. Among various LPWAN technologies, including **Long Range** (**LoRa**) [6, 7] and NB-IoT [8], LoRa stands out due

[1]This is an extension of the SenSys'22 article [1].

This publication was prepared with the support of a financial assistance award approved by the Economic Development Administration, Farms Food Future. It was also supported in part by NSF Grant #2239458, #2008837, a UC Merced Fall 2023 Climate Action Seed Competition grant, and a UC Merced Spring 2023 Climate Action Seed Competition grant. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.
Authors' Contact Information: Kang Yang, University of California Merced, Merced, California, United States; e-mail: kyang73@ucmerced.edu; Wan Du (Corresponding author), University of California Merced, Merced, California, United States; e-mail: wdu3@ucmerced.edu.

ACM 1550-4859/2024/07-ART98
https://doi.org/10.1145/3665928

to its operation in unlicensed sub-GHz frequency bands. Utilizing **Chirp Spread Spectrum** (**CSS**) modulation [9], LoRa enables sensor nodes to transmit data at low data rates (a few kbps) across several miles to gateways. However, during extended-distance transmissions, obstacles along the signal propagation path can adversely affect the **signal-to-noise ratio** (**SNR**) of LoRa packets, resulting in erroneous bits in the packets received by gateways [10].

In wireless networks, **Forward Error Correction** (**FEC**) codes serve to rectify bit errors by appending additional parity bits prior to each transmission [11]. LoRa utilizes Hamming codes, which are relatively simple to implement but known for their considerably sub-optimal error correction capacity [12]. The SemTech LoRa Design Guide quantifies the FEC coding gain under additive white Gaussian noise conditions [13]. Coding gain refers to the extent to which FEC codes can diminish the necessary SNR for successful data retrieval. The analysis reveals that Hamming codes yield only a marginal coding gain (less than 1 dB).

A variety of FEC codes, such as **Reed-Solomon** (**RS**) codes, Polar codes, and LDPC codes, have been extensively employed in contemporary wireless networks [11]. However, Polar and Turbo codes necessitate specialized hardware circuits for encoding data on the sender side, making them less suitable for LoRa networks [14, 15]. Additionally, the energy-intensive encoding process of RS codes, which involves computationally expensive and memory-intensive arithmetic calculations in **Galois Field** (**GF**), renders them unsuitable for LoRa networks as well [16]. In contrast, LDPC codes excel in error correction capability, even nearing the Shannon rate limit [17, 18]. They have found applications in 5G New Radio traffic channels [19], satellite communications [20], and the 802.11 WiFi protocol family [21]. Furthermore, LDPC encoding requires only straightforward XOR operations, allowing for implementation on sensor nodes with minimal energy consumption.

To reconcile the disparity between the limited FEC capacity of existing LoRa networks and the substantial coding gain offered by LDPC coding, we develop an efficient LDPC coding scheme for LoRa networks, *LLDPC*. Toward this end, we address the following three challenges.

Firstly, LDPC coding gain is reliant on its decoding algorithm, i.e., the **Soft Belief Propagation** (**SBP**) algorithm, which necessitates the **Log-Likelihood Ratio** (**LLR**) of each received bit as input. The LLR, a floating number, determines not only the bit value but also the confidence level associated with that value. Nevertheless, the CSS demodulation, as specified in LoRa, fails to provide any soft information pertaining to the output binary sequence. In *LLDPC*, we propose an innovative LLR extractor that leverages the amplitude spectrum of a symbol to compute the LLR of each bit within the symbol, where a symbol comprises a sequence of bits. The amplitude spectrum of a symbol is acquired through CSS demodulation. The LLR of a bit is calculated by comparing the amplitudes of frequency bins in the amplitude spectrum, where the corresponding bit value is either 1 or 0.

The second challenge is the low decoding efficiency due to the large LLR of some erroneous bits. Interference or ambient noise can result in packets containing erroneous bits after demodulation [22, 23]. These erroneous bits, with their large LLRs, can lead to the failure of the SBP algorithm. To address this issue, we attempt to decrease the LLR of erroneous bits by leveraging symbol-level information. Our key insight is that while we cannot identify the erroneous bits, it is relatively easy to identify the erroneous symbols. Erroneous symbols must encompass erroneous bits. Upon detecting an erroneous symbol, we promptly assign a low LLR to all its bits, significantly reducing the LLR of the erroneous bit and preventing a negative impact on LDPC coding gain. To achieve this, we must address two subsequent questions. *(1) how to identify erroneous symbols?* If a symbol is demodulated incorrectly, its amplitude spectrum comprises multiple high-amplitude frequency bins. By extracting a set of features that potentially characterize the correctness of demodulated symbols from the amplitude spectrum, we train a binary **Support Vector Machines**

(**SVM**) classifier [24]. *(2) By lowering the LLR of all bits in an identified erroneous symbol, we also reduce the LLR of some correctly-demodulated bits. Does this side effect influence the performance of LDPC coding?* Through empirical experiments, we ascertain that it does not adversely affect LDPC performance.

Thirdly, in accordance with the **Long Range Wide Area Network** (**LoRaWAN**) specification [25], the gateway is obliged to respond to the LoRa node with an **acknowledgment** (**ACK**) within one second. Nonetheless, the SBP algorithm necessitates an extensive amount of iterative updating operations, resulting in prolonged decoding latency. Recognizing that the parity-check matrix of LDPC codes can be converted into Tanner graphs, we seize the opportunity to execute LDPC decoding using **Graph Neural Networks** (**GNNs**). Consequently, we devise a GNN-based BP model for LDPC decoding [79]. Specifically, we employ GNN models to capture the relationship between original bits and parity-check bits. Experiments on a large-scale synthetic dataset are conducted to ascertain the optimal number of GNN layers. To train the GNN models end-to-end, we utilize binary cross-entropy loss.

Finally, we integrate the above LDPC coding scheme into the bit rate adaptation in LoRa networks. In LoRa, the bit rate is governed by the **Spreading Factor** (**SF**). When faced with a packet transmission failure, there are two potential ways: increasing the number of FEC parity-check bits (modifying the Coding Rate) or reducing the bit rate (altering the SF) [26]. To transmit a packet, we must decide the appropriate FEC **Coding Rate** (**CR**) for a given bit rate. To achieve this, we initially generate an SF-CR-SNR table through offline experimentation. This table documents the SNR thresholds for various SFs and CRs. The SNR threshold for a specific SF and CR pairing is derived from the corresponding BER-SNR curve, with a **Bit Error Rate** (**BER**) threshold of $1e^{-4}$. Based on the predicted channel SNR, we collaboratively search for the smallest SF and CR in the SF-CR-SNR table to transmit packets.

We implement *LLDPC* on **Universal Software Radio Peripheral** (**USRP**) N210 combined with a back-end host. The performance of *LLDPC* is assessed using both a large-scale synthetic dataset and an in-field testbed. Experimental results reveal that, in comparison to Hamming codes, *LLDPC* can augment the node lifetime by as much as 86.7%. Additionally, our GNN-based BP model diminishes the average decoding latency of the SBP algorithm by a factor of 58.09×.

In summary, this article makes the following contributions:

- *LLDPC* first integrates LDPC codes into LoRa networks, effectively enhancing the efficiency of data transmissions.
- We customize *LLDPC* to tackle a set of challenges, developing an LLR extractor to acquire the soft information of bits, implementing a symbol-aware LLR enhancement module to boost the capacity for error bit correction, and incorporating a GNN-based BP algorithm to decrease the LDPC decoding time.
- Through extensive in-field trials and simulations conducted with a large-scale synthetic dataset, *LLDPC* consistently demonstrates superior performance compared to conventional Hamming codes employed in LoRa networks. Additionally, we assess the robustness of the system by examining its sensitivity to distinct symbol features and diverse experimental configurations, thereby ensuring its dependability and versatility.

2 BACKGROUND AND MOTIVATION

This section presents the LoRa **physical layer** (**PHY**) and motivating experiments. Then, we discuss LDPC codes and LLR in the context of **M-ary quadrature amplitude modulation** (**M-QAM**).

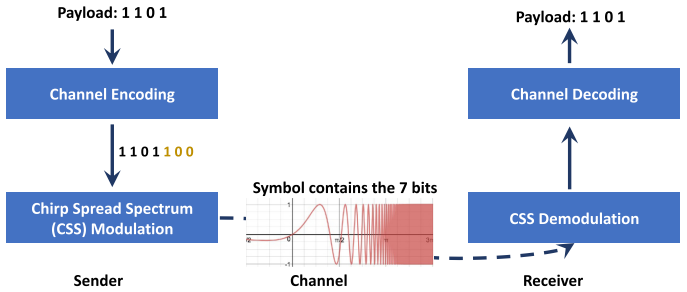


Fig. 1. The block-level overview of the LoRa PHY.

2.1 LoRa PHY

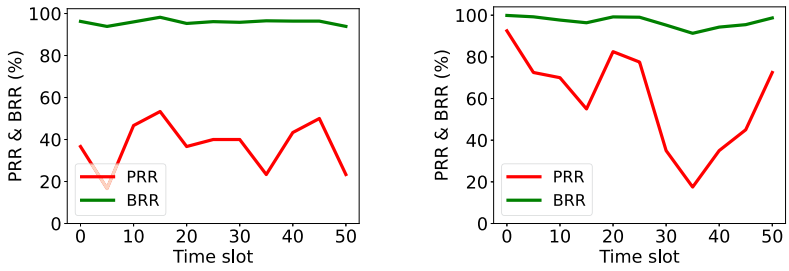
Figure 1 displays a simplified block diagram of a standard LoRa PHY [27], which includes channel encoding and modulation [28]. On the transmitter side, given a payload (e.g., "1 1 0 1"), channel encoding initially conducts a series of encoding procedures to enhance over-the-air resilience, encompassing FEC encoding, whitening, diagonal interleaving, and gray mapping. LoRa PHY employs Hamming codes as the default FEC coding scheme [13]. *LLDPC* aims at substituting Hamming codes with a more efficient FEC coding scheme in LoRa PHY. Subsequent to the aforementioned operations, CSS modulates the encoded data into multiple symbols. The number of bits in each symbol is determined by the SF (i.e., 7, 8, 9, and 10), enabling a symbol to represent an integer between 0 and $2^{SF} - 1$. A symbol is modulated by adjusting the initial frequency of a base chirp in increments of $BW/2^{SF}$, where BW denotes the channel bandwidth. A base chirp is a sinusoidal signal with a frequency that linearly increases from 0 Hz up to the channel bandwidth, e.g., 250 kHz [13].

On the receiver side, LoRa PHY executes demodulation and decoding. The demodulation process identifies a symbol's value by calculating its chirp's initial frequency. It multiplies the received signal by a down-chirp, whose frequency linearly decreases over time, and applies a **Fast Fourier Transform** (**FFT**) to the resulting signal. Consequently, the amplitude spectrum of each received symbol can be acquired. Every index of frequency bin correlates to a potential symbol value. The index of the frequency bin with the highest amplitude is used to determine the value of a symbol. The bits from all recognized symbols in a packet are combined into a binary sequence, which is further processed through Gray demapping, deinterleaving, and dewhitening in order. Ultimately, the original payload can be retrieved through FEC decoding.

2.2 Motivating Experiments

We investigate the **Packet Reception Ratio** (**PRR**) and **Bit Reception Ratio** (**BRR**) of LoRa links under two CRs of Hamming codes, i.e., 4/5 and 4/7. For the 4/5 CR, a one-bit parity check is appended to every four bits, enabling the detection of a single-bit error within the five-bit group, albeit without correction capabilities. Conversely, the 4/7 CR allows for the correction of one erroneous bit within each seven-bit segment through Hamming codes. LoRa nodes, as depicted in Figure 22, periodically transmit 32-byte packets to the gateway using an SF of 10, a transmission power of 14 dB, a bandwidth of 125 kHz, and a frequency channel of 904.3 MHz. The duration of the experiment is set to 2.5 hours. PRR and BRR are evaluated at three-minute intervals.

Figure 2 illustrates the PRR and BRR for LoRa links across two CRs. The observed high BRR in conjunction with a low PRR suggests that, although a significant portion of packets are corrupted during transmission, the total number of erroneous bits within these packets remains relatively low. This phenomenon aligns with findings reported in [12]. Given that Hamming codes with CRs



(a) Coding rate: 4/5. (b) Coding rate: 4/7.

Fig. 2. PRR and BRR for two different CRs.

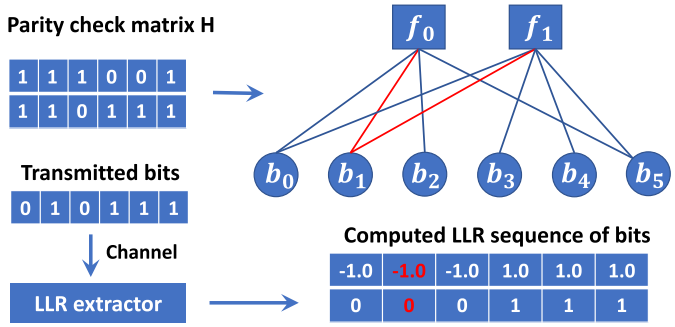


Fig. 3. The top illustrates a parity-check matrix and its associated Tanner graph. In $\mathcal{H}$, each row represents a check node (square) while each column signifies a bit node (circle) in the corresponding graph on the right. The bottom is an example of the computed six-bit LLR using our LLR extractor.

of 4/5 and 4/7 lack the capability to correct even this modest quantity of errors, we are prompted to develop a more effective FEC coding scheme for LoRa networks, such as LDPC codes.

2.3 Low-Density Parity-Check Codes

LDPC codes encode a data payload of K bits into a packet of N bits by appending M parity check bits to the K payload bits, where $M = N - K$ and the CR equals K/N. A sparse parity-check matrix $\mathcal{H}_{M\times N}$ can be generated randomly to obtain an (N, K) LDPC code [17, 29]. The Tanner graph [30] represents $\mathcal{H}_{M\times N}$, as illustrated in Figure 3. Nodes denoted by f_i are check nodes, while those identified by b_j are bit nodes. Each row in $\mathcal{H}$ signifies a check node constraint, i.e., the XOR sum of participating bit nodes equals zero. If $h_{ij} = 1$, this implies that the bit node (b_j) participates in the constraint of the check node (f_i). The matrix $\mathcal{H}$ is generated offline. The sender can effortlessly produce the encoded data using simple XOR operations with the matrix $\mathcal{H}$. Hence, the LDPC encoding process is computationally light and can be executed on LoRa nodes, such as the Arduino Uno board [31] in our implementation.

LDPC Decoder. Two primary LDPC decoding algorithms exist: Bit-flipping [32] and SBP [33]. Bit-flipping operates as a hard-decision decoding algorithm, utilizing a binary bit stream as input for decoding the data. Conversely, SBP functions as a soft-decision decoding algorithm, accounting for the reliability of received bits by using LLR as input to generate improved estimates. Soft-decision decoding surpasses hard-decision decoding in performance by 2.5 dB SNR[34]. In Section 6.5, our experiments further demonstrate the limited performance of Bit-flipping.

Consequently, this article primarily focuses on SBP. The SBP decoding algorithm can be outlined as follows [33]:

— *Step 0: The First Message from Bit Nodes to Check Nodes.* When a packet is received, the LLR of each bit can be obtained by demodulation. To initialize the decoding process, each bit node sends its LLR to its connected check nodes.

— *Step 1: Updating Messages Sent from Check Nodes to Bit Nodes.* After a check node f_i receives the messages from all its connected bits nodes, it calculates the message that will send back to bit node b_j as follows:

$$\eta_{f_i \to b_j} = 2 \tanh^{-1} \left(\prod_{b'_j \in N(f_i) \setminus b_j} \tanh \left(\frac{\lambda_{b'_j \to f_i}}{2} \right) \right), \tag{1}$$

where $N(f_i)$ is the set of bit nodes connected to check node f_i, and $\lambda_{b'_j \to f_i}$ is equal to LLR(b'_j) for the first iteration.

— *Step 2: Updating Messages Sent from Bit Nodes to Check Nodes.* After a bit node b_j receives the messages from all its connected check nodes, it prepares the message that will be sent back to check node f_i,

$$\lambda_{b_j \to f_i} = \text{LLR}\left(b_j\right) + \sum_{f'_i \in M(b_j) \setminus f_i} \eta_{f_{i'} \to b_j}, \tag{2}$$

where $M\left(b_j\right)$ is the set of check nodes connected to bit node b_j.

— *Step 3: Verifying Termination Condition.* Before all bit nodes send the updated messages to their connected check nodes, they first verify whether the termination conditions are met. To do so, every bit node b_j updates its LLR value λ_{b_j}, according to the messages $\eta_{f'_i \to b_j}$ received from all its connected check nodes,

$$\lambda_{b_j} = \text{LLR}\left(b_j\right) + \sum_{f'_i \in M(b_j)} \eta_{f'_i \to b_j}. \tag{3}$$

Let y_{b_j} denote the output of the SBP algorithm. The SBP slices λ_{b_j} to determine the decoded output bit y_{b_j}, i.e., if $\lambda_{b_j} \geq 0$, then $y_{b_j} = 1$; otherwise, $y_{b_j} = 0$. We can obtain the binary sequence $\mathbf{y} = [y_{b_0}, y_{b_1}, \ldots, y_{b_{N-1}}]$. The SBP algorithm stops if $\mathbf{y} \cdot \mathcal{H}^T = 0$ or if the maximum number of iterations is reached; otherwise, the algorithm starts another iteration from Step 1.

2.4 LLR in M-QAM

LLR is essential for the SBP algorithm and can be computed in M-QAM demodulation, a technique commonly employed in contemporary wireless networks such as WiFi and 5G [35, 36]. The calculation proceeds as follows:

$$\text{LLR}\left(b_j\right) = \log \frac{\mathcal{F}\left(b_j = 1 \mid \mathbf{r}\right)}{\mathcal{F}\left(b_j = 0 \mid \mathbf{r}\right)}, \text{ where } 0 \leq j < log2(M), \tag{4}$$

where $\mathbf{r}$ represents a received symbol and b_j refers to the jth bit in the symbol $\mathbf{r}$.

Upon receiving a symbol $\mathbf{r}$, it is depicted as a point on the constellation diagram (I-Q plane), with the M standard symbols occupying fixed positions. For instance, in Figure 4, the cross symbol indicates the received symbol, while the other 16 circle symbols represent the standard symbols. Subsequently, the Euclidean distances between the received symbol $\mathbf{r}$ and the M standard symbols in the constellation diagram are determined. Ultimately, $\mathcal{F}\left(b_j = 1 \mid \mathbf{r}\right)$ is computed as the sum of the Euclidean distances between the received symbol $\mathbf{r}$ and the $M/2$ standard symbols with the jth

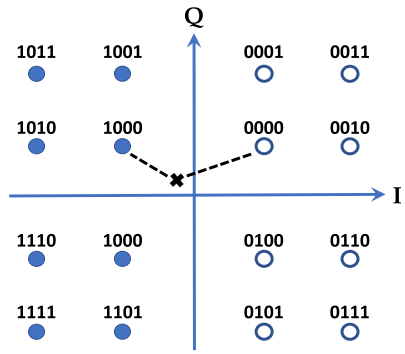


Fig. 4. The illustration of LLR calculation for 16-QAM.

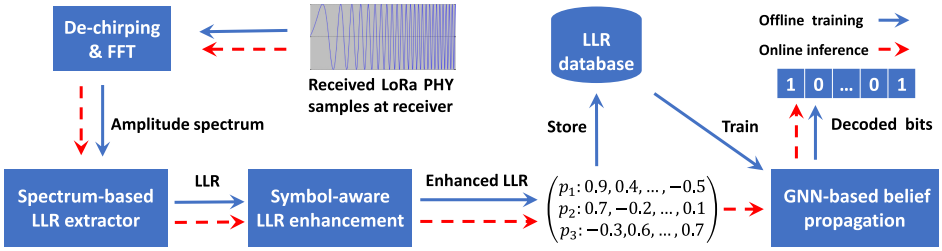


Fig. 5. The overall architecture of *LLDPC*.

bit equal to 1. Analogously, $\mathcal{F}\left(b_j = 0 \mid \mathbf{r}\right)$ amounts to the sum of the Euclidean distances between the symbol $\mathbf{r}$ and the remaining $M/2$ standard symbols with the jth bit equal to 0.

Based on Equation (4), LLR can be employed to ascertain bit values, meaning if $LLR(b_j) \geq 0$, then $b_j = 1$; otherwise, $b_j = 0$. Additionally, LLR can supply the confidence level for bit b_j being either 1 or 0. A high absolute LLR value indicates that the received symbol is closer to the standard symbol with the jth bit as 1 or 0.

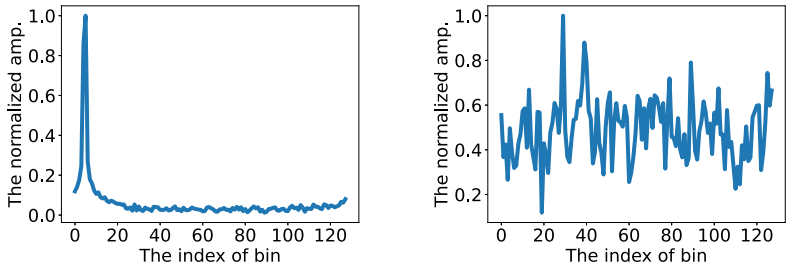
Challenges in Implementing LDPC in LoRa: Firstly, LoRa utilizes CSS modulation, which differs from M-QAM modulation. No existing methods can calculate the LLR of received bits during CSS demodulation. The second challenge involves the low decoding efficiency resulting from large LLRs of some erroneous bits. These bits may possess large LLRs, leading to the failure of the SBP algorithm. Moreover, SBP necessitates numerous iterations to achieve effective error correction, causing extended decoding latencies (5.46 seconds in our Raspberry Pi 3 single-board computer implementation). However, according to the LoRaWAN specification [25], a gateway must respond with an ACK to the LoRa node within one second. Although parallel LDPC decoding implementations exist [37], they demand high-end hardware or quantum computing platforms, rendering them unsuitable for low-cost, large-scale LoRa networks.

3 DESIGN OF *LLDPC*

We present the design of *LLDPC* and its three components: the spectrum-based LLR extractor, the symbol-aware LLR enhancement module, and the GNN-based belief propagation algorithm.

3.1 Overview

Figure 5 depicts the architecture of *LLDPC*. Upon receiving a signal from a sensor node, a LoRa gateway initially acquires the amplitude spectrum of the received symbols through de-chirping



(a) Correctly demodulated symbol. (b) Incorrectly demodulated symbol.

Fig. 6. The amplitude spectrum obtained from CSS demodulation for two symbols at different SNRs.

and FFT operations. Utilizing the spectrum results of each symbol, our LLR extractor calculates the LLRs for every bit in the symbol (Section 3.2). To optimize SBP decoding efficiency, we fine-tune the LLR of all bits by employing symbol-level information (Section 3.3). Ultimately, *LLDPC* feeds the LLR sequence into the GNN model for LDPC decoding (Section 3.4). The decoding latency remains low (under 2 seconds), enabling the gateway to transmit an ACK packet to the sensor node within the LoRaWAN standard's specified ACK constraint (2 seconds).

Both the SVM model for the symbol-aware LLR enhancement module and the GNN model are trained offline. We gather training data from a large-scale synthetic dataset created through experiments on the USRP N210 platform. In our experiments, since we record the data payloads transmitted by the sensor nodes, we use them to label the received packets at the gateway.

3.2 Spectrum-Based LLR Extractor

3.2.1 CSS Demodulation. After conducting de-chirping and FFT, the amplitude spectrum of a received symbol can be obtained. The index of the frequency bin with the maximum amplitude is chosen as the symbol's value. This CSS demodulation process can be regarded as a classification task, which yields the probability of all frequency bins. By performing the SoftMax operation [38] on the amplitude spectrum, the probability of each frequency bin can be computed:

$$\mathcal{P}(t = \text{bin}_x) = \frac{\exp\left(A_{\text{bin}_x}\right)}{\sum_{y\in[0,\ 2^{SF}-1]} \exp\left(A_{\text{bin}_y}\right)}, \tag{5}$$

where t represents the modulated value of a transmitted symbol, exp denotes the exponential function, and $\mathcal{P}(t = \text{bin}x)$ signifies the probability that the modulated value equals the xth frequency bin. The $A\text{bin}_x$ refers to the amplitude of the xth bin.

The index of the frequency bin with the highest probability is selected as the demodulation result. If the chosen probability (i.e., amplitude) is substantially greater than the others, the classification result's confidence is higher, as illustrated in Figure 6(a) for the correctly demodulated symbol. Conversely, if the highest amplitude in the amplitude spectrum closely resembles the amplitude of other frequency bins, determining the symbol's value becomes difficult, as demonstrated in Figure 6(b). Nonetheless, LoRa CSS demodulation merely picks the frequency bin with the highest amplitude as the output, disregarding the confidence information.

3.2.2 LLR Calculation. Each symbol consists of SF bits. Let $S^+_{b_j}$ denotes the set of symbols with the jth bit equal to 1, and $S^-_{b_j}$ represents the set of symbols with the jth bit equal to 0 ($0 \le j < SF$). For instance, if SF is 3, then $S^+_{b_0} = 4,\ 5,\ 6,\ 7$ and $S^-_{b_0} = 0,\ 1,\ 2,\ 3$. The LLR of the jth bit in a symbol

can be computed as follows:

$$\mathrm{LLR}\left(b_j\right) = \log \frac{\sum_{t \in S_{b_j}^{+}} \mathscr{P}\left(t \mid \tilde{r}\right)}{\sum_{t \in S_{b_j}^{-}} \mathscr{P}\left(t \mid \tilde{r}\right)} \approx \log \frac{\max_{t \in S_{b_j}^{+}} \mathscr{P}\left(t \mid \tilde{r}\right)}{\max_{t \in S_{b_j}^{-}} \mathscr{P}\left(t \mid \tilde{r}\right)}, \tag{6}$$

where $\mathscr{P}\left(t \mid \tilde{r}\right)$ denotes the probability that the transmitted symbol's value is t given the received signal $\tilde{r}$. It is calculated using Equation (5) based on the symbol's amplitude spectrum. Thus, Equation (6) specifies that $\mathrm{LLR}\left(b_j\right)$ is the ratio of the sum of probabilities that all symbols in set $S_{b_j}^{+}$ concur on $b_j = 1$ to the sum of probabilities that all symbols in set $S_{b_j}^{-}$ concur on $b_j = 0$.

We approximate the sum of probabilities as the highest probability for two reasons. First, when SF is large (e.g., SF10), each set ($S_{b_j}^{+}$ and $S_{b_j}^{-}$) contains 512 items. Calculating the probability of each symbol and the sum of all probabilities in each set is time-consuming. Second, the summation can be distorted by a single outlier, while the maximization operation is more resilient. This approximation is also commonly used in existing wireless networks [36]. We conduct experiments to assess the difference between the sum and max operations. For each packet, we calculate the LLR using sum and max, respectively. We then measure the absolute difference between the LLR calculated by sum and max for the corresponding bits. The average difference in LLR is only 0.13, indicating a negligible impact on LLR calculation.

The LLR can be utilized to identify the values of bits. A positive LLR indicates a bit value of 1, while a negative LLR signifies a bit value of 0. The experiments in Section 6 reveal that our LLR extractor produces an identical binary sequence to that of LoRa CSS demodulation. Consequently, our LLR extractor does not affect the BER of received packets. For instance, if the original bit is 1 but the CSS demodulation produces 0, our LLR extractor will yield a negative LLR. Bit errors occur due to interference or environmental noise during wireless transmission and will be corrected by the following FEC decoding.

3.3 Symbol-Aware LLR Enhancement Module

In this section, we first explore the influence of the calculated LLR of erroneous bits on the SBP algorithm's efficiency. Subsequently, we introduce a symbol-aware LLR enhancement module that incorporates symbol-level information to refine the LLR of some bits.

3.3.1 The Decoding Efficiency of SBP and the LLR of Erroneous Bits. Figure 3 illustrates a simple scenario where all bit LLRs are normalized, and they are assumed to have the maximum absolute values, i.e., 1.0. There are six transmitted bits. The parity check matrix and its corresponding Tanner graph are also displayed in Figure 3. We assume the second bit is in error, meaning the second bit is flipped from 1 to 0.

We will now demonstrate how SBP attempts to correct the second bit during the decoding iteration. In the first iteration, after bit nodes send their initial LLRs to the connected check nodes, SBP updates the check node message.

$$\begin{aligned} \eta_{f_0 \to b_1} &= 2 \tanh^{-1}\left(\prod_{b_j \in [0,\ 2,\ 5]} \tanh\left[0.5 LLR\left(b_j\right)\right]\right) = 0.198 \\ \eta_{f_1 \to b_1} &= 2 \tanh^{-1}\left(\prod_{b_j \in [0,\ 3,\ 4,\ 5]} \tanh\left[0.5 LLR\left(b_j\right)\right]\right) = -0.091 \end{aligned}. \tag{7}$$

This equation is instanced from Equation (1). Both messages are sent to bit node b_1. Considering these two messages and its current LLR, bit node b_1 updates its LLR to -0.893 using Equation (3).

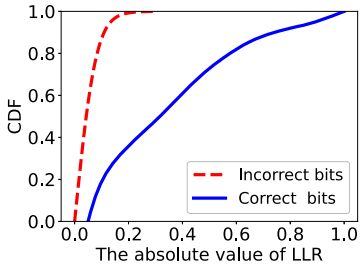


Fig. 7. LLR for correct and erroneous bits w/o symbol-level info.

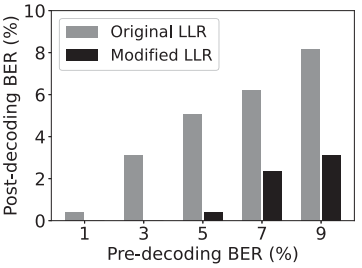


Fig. 8. Post-decoding BER by SBP using original and enhanced LLRs.

After this iteration, the LLR of the second bit decreases from -1.0 to -0.893. Despite running 1,000 iterations, this value eventually converges to -0.77, which means the second bit remains 0. Therefore, the SBP algorithm is unable to correct this erroneous bit, even with numerous iterations, due to its large absolute LLR value.

From this example, it is evident that the SBP's efficiency is impacted by the large LLR of erroneous bits. We further examine the extent of the large absolute LLR values for erroneous bits by plotting the CDF of the absolute LLR values for correct and erroneous bits based on a synthetic dataset collected in Section 6. Figure 7 reveals that the absolute LLR values of erroneous bits can reach up to 0.32, which negatively affects the SBP's efficiency.

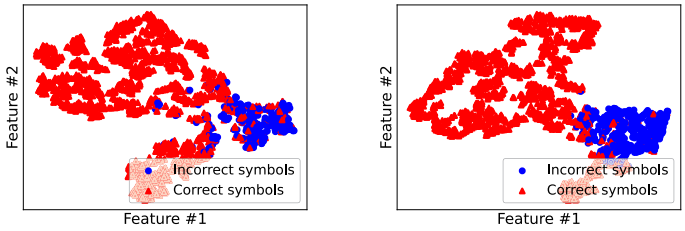
At the same time, we investigate the decoding efficiency of SBP when the initial LLR of the second bit is manually adjusted from -1.0 to -0.1. SBP follows the same update process, and we discover that after two iterations, the LLR of the second bit changes from -0.1 to 0.007, allowing SBP to successfully correct errors. By reducing the absolute LLR value of erroneous bits, we can enhance the error correction efficiency of SBP.

In order to further validate the aforementioned observation, we conduct a simulation experiment. The LLRs of all bits are generated randomly based on their bit values; for instance, if a bit value is 1, its LLR will be a positive floating-point number. Erroneous bits are introduced with a specific pre-decoding BER. We use the term "pre-decoding BER" to denote the BER before LDPC decoding and "post-decoding BER" for the BER after LDPC decoding. Figure 8 demonstrates that when the absolute LLR value is similar to that of correct bits, SBP is unable to correct erroneous bits, even if the BER is low (depicted by the gray bar). Conversely, by adjusting the absolute LLR value of erroneous bits to a smaller floating-point number, SBP exhibits enhanced error correction capabilities (indicated by the black bar).

However, identifying erroneous bits presents a considerable challenge. Fortunately, we can indirectly detect erroneous bits by examining symbol-level information. When a bit is incorrect, the symbol containing that bit must be demodulated erroneously. If we can predict that a symbol has been demodulated incorrectly, we can assign a low LLR to all bits within that symbol. Hence, these erroneous bits will possess a lower LLR, significantly enhancing the decoding efficiency of SBP.

3.3.2 Erroneous Symbol Detection. We propose symbol features to classify the correctness of symbols using an SVM model.

Feature. To discover meaningful features related to symbol correctness, we examine the distribution of the amplitude spectrum. Figure 6 reveals that when a symbol is demodulated incorrectly, the spectrum consists of multiple high-amplitude frequency bins relative to the highest-amplitude bin. We quantify these high-amplitude bins by calculating the amplitude ratio between the highest amplitude and their respective amplitudes. Specifically, frequency bins are sorted by their amplitudes, and we compute the amplitude ratios of the top-$\mathcal{S}$ high-amplitude bins



(a) When SF = 7. (b) When SF = 10

Fig. 9. The representation visualization with t-SNE for our proposed symbol features when SF = 7/10.

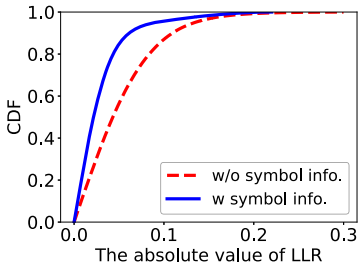


Fig. 10. Absolute LLRs for error bits w/o and w/ symbol-level info.

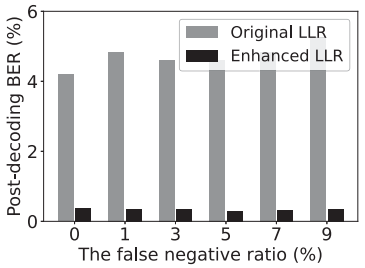


Fig. 11. Impact of enhanced LLRs for correct bits with different FNR.

relative to the highest-amplitude bin. These ratios form a feature vector of length $\mathcal{S}$. Empirically, we set $\mathcal{S}$ as five in our experiments.

To assess the effectiveness of the proposed features, we utilize the **t-distributed Stochastic Neighbor Embedding (t-SNE)** [39] technique for visualization. As depicted in Figure 9, symbols display a strong clustering effect for SF7 or SF10. Therefore, our proposed features can be employed for binary symbol classification, predicting whether symbols are demodulated correctly or not.

Classifier. To achieve this objective, we employ an SVM classifier [24] with a **radial basis function (RBF)** kernel for symbol classification. As demonstrated in Section 6.5, our SVM model can predict symbol correctness with a false negative ratio of 4.2% and a false positive ratio of 0.3%. We opt against deep learning-based approaches for erroneous symbol detection due to the low dimensionality of our features. Deep learning methods are typically designed to handle high-dimensional input features [40]. In contrast, tree-based methods excel when working with diverse data types, such as continuous and discrete features.

3.3.3 The Calculation of Enhanced LLR. After classifying symbol correctness, we augment the LLR of bits associated with symbols predicted as incorrectly demodulated. We multiply the probability of a symbol being correctly demodulated by the original LLR of bits, a probability obtained from the SVM classifier. This reduces the absolute LLR value of erroneous bits. Figure 10 displays the CDF of absolute LLR before and after integrating symbol-level information, indicating a reduction in the absolute LLR values of erroneous bits.

Simultaneously, we also decrease the absolute LLR values of some correct bits. Does this compromise the performance of the SBP algorithm? We conduct a simulation similar to that in Figure 8, with the difference being that we not only reduce the absolute LLR value of erroneous bits but also randomly reduce the absolute value of correct bits with varying false positive ratios (FNR). Figure 11 reveals that the reduced LLRs of correct bits have minimal impact on SBP decoding performance. The reasoning behind this is that lowering LLR for correct bits does not adversely affect

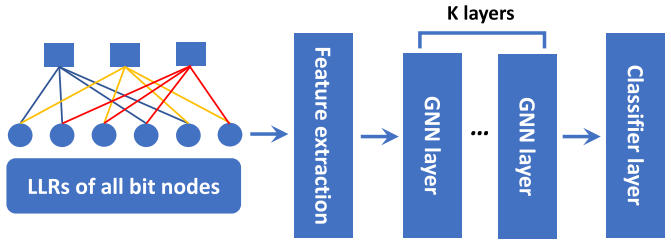


Fig. 12. The architecture of the GNN-based BP algorithm.

SBP updating. However, by assigning a small LLR to erroneous bits, their significant influence on SBP can be substantially mitigated.

3.4 GNN-Based Belief Propagation Algorithm

The GNN-based model has proven effective in handling graph-structured data, capturing pairwise dependencies between variables, and propagating information throughout the graph [41]. Moreover, the GNN model generalizes to capture higher-order constraints between bit nodes and check nodes on factor graphs, while also parameterizing the SBP algorithm [41]. Motivated by these recent advances in GNNs, we employ the GNN model for efficient LDPC decoding on Tanner graphs.

Figure 12 illustrates the architecture of GNN-based belief propagation. The Tanner graph and enhanced LLRs of each bit are fed to the feature extraction module. This module extracts five matrices that serve as input to the GNN layers. These GNN layers can learn high-order constraints among nodes and parameterize the SBP algorithm. Finally, a classifier layer is used to obtain the final bit values.

Feature Extraction. This module is responsible for generating node features and edge features. Node features consist of two types: bit node features and check node features. The initial LLR of the bit nodes is employed as the bit node feature. For check node features, we concatenate the features of all connected bit nodes as the check node feature. Edge features are acquired by concatenating the bit node feature and check node feature if an edge exists between the bit node and check node. Thus, we generate two matrices: the bit node matrix and the check node matrix. The bit node matrix indicates the check nodes each bit node is connected to, while the check node matrix specifies the bit nodes each check node is linked to. These two matrices represent the structure of the Tanner graph.

GNN Layer. The GNN layer updates the bit node features based on node features, edge features, and graph structure.

$$\tilde{\mathbf{v}}_{b_j} = \sum_{p_i \in M(b_j)} \mathcal{Q}_1\left(\mathbf{e}_{p_i \to b_j}\right) \mathcal{Q}_2\left(\mathbf{f}_{p_i}, \mathbf{v}_{b_j}\right), \tag{8}$$

where $\mathcal{Q}_1$ maps edge features $\mathbf{e}p_i \to b_j$ to a weight matrix, and $\mathcal{Q}_2$ maps check node feature $\mathbf{f}p_i$ and bit node feature $\mathbf{v}b_j$ to a feature vector, as illustrated in Figure 13. An updated bit node feature $\tilde{\mathbf{v}}_{b_j}$ can then be generated through matrix multiplication and summation operations.

In this layer, the number of GNN layers $\mathcal{K}$ must be chosen. We conduct experiments to empirically determine the value of $\mathcal{K}$. We randomly select 75% of the dataset to train GNN models with varying GNN layers on a local PC offline. The remaining 25% of the dataset is used to test the post-decoding BER of the GNN layer, while decoding latency is measured on both a local PC and a single-board computer (Raspberry Pi 3). Table 1 demonstrates the effect of $\mathcal{K}$ on decoding

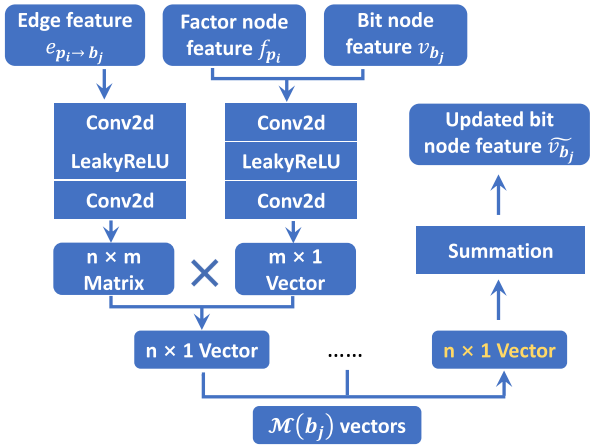


Fig. 13. The architecture of the GNN layer.

Table 1. Effect of the Number of Layers $\mathcal{K}$ on the Post-Decoding BER and Decoding Latency with SNRs in the Range of $[-30, -5]$ dB

# of layers $\mathcal{K}$	5	**8**	12
BER (%)	23.09 ± 21.83	23.06 ± 21.76	23.02 ± 21.05
Latency (s)	0.009/0.207	0.021/0.434	0.054/1.404

The latency is measured from two platforms, i.e., PC/Raspberry Pi 3.

performance. It can be observed that the choice of $\mathcal{K}$ has a substantial impact on decoding latency, but less on decoding accuracy. To achieve a satisfactory performance and acceptable decoding latency, we set the number of layers $\mathcal{K}$ to 8.

Classifier Layer. Finally, the updated bit node feature $\tilde{\mathbf{v}}_{b_j}$ is passed to the classifier layer to determine the bit value, either 0 or 1.

4 MODEL TRAINING FOR *LLDPC*

In this section, we describe the training details, including the hardware implementation of LoRa nodes and gateways, and the offline training of LLDPC models. Two models in *LLDPC* are trained, namely the SVM model for symbol-aware LLR enhancement (Section 3.3) and the GNN-based BP module (Section 3.4). *LLDPC* is implemented on a local computer equipped with an Intel(R) Core (TM) i9-11900KF @ 3.50 GHz CPU with 16 cores. A **graphics processing unit (GPU)** card (NVIDIA GEFORCE RTX 3080 Ti) is employed to accelerate the training process of GNN modules.

Implementation. LoRa nodes are custom-built using the SX1276 Radio [42] on Arduino Uno host boards [31]. We employ the USRP N210 **software-defined radio (SDR)** platform to capture over-the-air LoRa signals, operating on a UBX daughter board at the 904.3 MHz bands. The sampling rate is set to 1 MHz. The captured signal samples are subsequently transmitted to a back-end host for demodulation using the methods proposed in [43].

Due to hardware constraints, we are unable to disable Hamming codes. As an alternative, on the sender side, we encode a 32-byte payload data into a 40-byte encoded data using LDPC encoding (CR is 4/5). The 40 bytes of data are then transmitted to the receiver through modulated symbols after successive Hamming encoding, whitening, diagonal interleaving, and gray mapping in the sender's hardware. Regardless of the LDPC CR, the Hamming codes CR is consistently set to 4/5.

For CR of 4/5, every four bits are concatenated with one parity-check bit. With a 4/5 CR, Hamming codes can detect one erroneous bit but are unable to correct any error in the five bits.

At the receiver side, we use the amplitude spectrum of demodulated symbols to compute the LLR of all bits, which include the encoded data from LDPC and Hamming codes. Subsequently, we perform Gray demapping, deinterleaving, and dewhitening sequentially on the calculated LLR. We discard one parity-check bit and extract only the LLR of the four data bits for every five bits (Hamming codes' CR is 4/5). This approach removes the influence of Hamming codes on LDPC decoding performance. All these operations are implemented using a publicly available GitHub library [43, 44]. As a result, we obtain the LLR with 40 bytes of data bits encoded by LDPC codes. A symbol-aware LLR enhancement module enhances these LLRs. Lastly, we execute the GNN-based BP algorithm to perform LDPC decoding.

Parity Check Matrix. In this article, we employ *regular* LDPC codes, characterized by a constant number of "1" in each row and column of parity check matrices. We define parity check matrices of varying dimensions for different CRs. The candidate CRs include 4/5, 4/6, and 4/7. Given a payload size of 32 bytes, the corresponding dimensions of the parity check matrices are 64×320, 128×384, and 192×448.

Training Data Collection. We gather LoRa I/Q signals using the USRP N210 to create training datasets. LoRa nodes periodically transmit random packets from five locations within an office building. We collect 12,000 packets at high SNR (>20 dB) with diverse transmission settings, encompassing 4 SFs (i.e., 7, 8, 9, 10) and 3 CRs (i.e., 4/5, 4/6, 4/7). Utilizing the collected high SNR packets, we employ data augmentation to generate 3.6 million packets with SNR values ranging from −35 dB to −5 dB in 0.1 dB increments. To generate new LoRa packets with specific SNR values, we introduce Gaussian white noises with corresponding amplitudes to the collected I/Q samples, a well-established data enhancement technique [45].

Training SVM model. To train the SVM model, we first extract the feature vectors of symbols from the amplitude spectrum. Through experimentation, we set the feature vector length to 5. The symbol label can be determined by comparing the demodulated value of the symbol to its actual value. The SVM models employ the RBF as the kernel. One parameter, $\mathcal{C}$, must be determined for the RBF kernel. Parameter $\mathcal{C}$ balances the misclassification of training examples against the simplicity of the decision surface. A low $\mathcal{C}$ yields a smooth decision surface, while a high $\mathcal{C}$ aims at classifying all training examples accurately. Empirically, $\mathcal{C}$ is set to 1.0. Moreover, considering the imbalanced number of two classes, we designate the field of *classWeight="balanced"* to automatically adjust the sample weight in inverse proportion to the number of classes.

Training GNN model. The binary cross-entropy loss $\mathcal{L}$ for the GNN model is derived from the predicted LLR of all bit nodes $\hat{p}(\mathbf{c})$ and ground truths $\mathbf{c}$. The ground truths $\mathbf{c}$ are the transmitted bits known to the LoRa receiver during the training phase. Upon calculating the loss, we propagate the loss back through the network to update the GNN modules.

$$\text{Loss}(\Theta) = \mathcal{L}\left(\mathbf{c}, p(\mathbf{c})\right), \tag{9}$$

where Θ represents all the parameters of the GNN-based BP algorithm. We train the GNN model using the enhanced LLRs and the labels for each bit. We design a GNN model comprising eight GNN layers. Each layer shares identical Q_1 and Q_2 functions, which are realized as a two-layer **Multilayer Perceptron** (**MLP**) network structured as follows: MLP (64) - MLP (4). The first layer employs a ReLU activation function, while the second layer operates without an activation function. The model is implemented using PyTorch [46] and trained with the Adam optimizer, employing an initial learning rate of 0.01. After every 10,000 samples, the learning rate is reduced by a factor of 0.98. The GNN model parameters are uniformly initialized within the range $[-0.1, 0.1]$. The batch size is set to 32.

Table 2. The SNR Threshold (dB) Under Different SFs and CRs for *LLDPC*, where the SNR Threshold with CR of 4/4 is obtained from LoRa Standards [13]

SF	CR	SNR threshold	SF	CR	SNR threshold
7	4/4	−7.5	9	4/4	−12.5
7	4/5	−8.7	9	4/5	−14.3
7	4/6	−9.4	9	4/6	−15.8
7	4/7	−10.6	9	4/7	−16.5
8	4/4	−10.0	10	4/4	−15.0
8	4/5	−11.2	10	4/5	−16.8
8	4/6	−12.7	10	4/6	−18.3
8	4/7	−14.0	10	4/7	−19.5

5 INTEGRATION OF *LLDPC* INTO LORAWAN

In the preceding two sections, we have devised our LDPC decoding strategy for the LoRa receiver side. In this section, we further introduce a mechanism to jointly configure SFs and CRs for LoRa transmitters, taking into account our *LLDPC*-based error correction system.

LoRaWAN [25, 47] employs the **Adaptive Data Rate** (**ADR**) algorithm [48] to select the appropriate SF for each sender node. Initially, it estimates the link's SNR by averaging the SNRs of several recently received packets. Subsequently, it selects the highest data rate with an SNR threshold lower than the estimated SNR.

The potential CRs include 4/5, 4/6, and 4/7. In *LLDPC*, LDPC codes with distinct CRs for a single SF can also offer various SNR thresholds, as demonstrated in Table 2. Consequently, we need to jointly set SFs and CRs. We establish the SNR threshold corresponding to a BER of $1e^{-4}$, as this BER can achieve a 99.68% packet delivery rate for 40-byte packets.

Upon receiving a packet from a LoRa node, the gateway predicts the SNR for the following transmission by computing the weighted average of the three most recently received packets. Based on the predicted SNR, we consult Table 2 to identify the candidate SFs and CRs with an SNR threshold lower than the predicted SNR. The final SF and CR are selected based on the shortest transmission time, as determined by the LoRa standard [13]. The updated SF and CR are then transmitted back to the LoRa node. The LoRa node utilizes the most recent SF and CR to transmit the subsequent packet.

Disabling LDPC Codes. The CR of 4/4 is also incorporated in Table 2. Utilizing LDPC codes invariably introduces overhead from parity check bits. Hence, we must consider the CR of 4/4. If the CR is 4/4, LDPC codes will not be employed. For instance, if the predicted SNR exceeds −7.5 dB, we can use SF7 and a CR of 4/4, as the predicted SNR surpasses the SNR threshold of SF7 and a CR of 4/4, which offers the shortest transmission time among all available configurations. In this situation, LDPC codes are not necessary.

6 EVALUATION

We perform comprehensive experiments to assess *LLDPC*'s efficacy on large-scale synthetic datasets and in-field experimental scenarios.

Evaluation on Synthetic Datasets. The synthetic dataset facilitates the fine-grained manipulation of SNRs. This allows for a thorough evaluation of *LLDPC*'s performance under a wide array of conditions, encompassing all possible SFs, CRs, and SNRs. Moreover, it ensures a sufficient quantity of data for training GNN models. Section 6.2 initially discusses the general performance of *LLDPC*

on the extensive synthetic dataset, followed by an examination of *LLDPC*'s performance under distinct experimental configurations in Sections 6.3 and 6.4. Furthermore, we assess the effectiveness of the proposed components in Section 6.5.

In-Field Experiments. Considering that the synthetic dataset encompasses solely synthetic Gaussian white noise in indoor settings, we proceed to examine *LLDPC* in a campus environment. Thus, in-field experiments serve to evaluate *LLDPC*'s performance in the presence of additive white Gaussian noise and additional noise types, such as multipath interference. Section 6.7.1 scrutinizes the performance of *LLDPC* at each location in in-field experiments. So far, we have assessed the performance of *LLDPC* under separate SFs and CRs. Subsequently, we will analyze *LLDPC*'s performance when integrated with LoRaWAN. In Section 6.7.2, our system selects SFs and CRs to accommodate link quality. Lastly, we quantify the overhead of *LLDPC* in Section 6.8.

6.1 Experimental Setup

6.1.1 Performance Criteria. We utilize three metrics to evaluate the performance of *LLDPC*.

Bit Error Ratio (BER). This metric indicates the fraction of incorrect bits over the total number of bits in a packet. The BER is determined individually for each transmitted packet.

Packet Delivery Rate (PDR). It characterizes the proportion of packets transmitted by the sender compared to the volume of packets received by the gateway.

Lifetime. The node lifetime represents the time span from its initial activation to the depletion of its battery. This metric evaluates the energy efficiency of *LLDPC*. The lifetime is computed using Equation (10), as found in [49–51].

$$\text{Lifetime} = T_{\text{cycle}} \cdot \frac{E_{\text{battery}}}{E_{\text{cycle}}}, \tag{10}$$

where T_{cycle} denotes the duration of a single sensing cycle, such as 10 minutes. E_{cycle} is the energy expended by the LoRa nodes throughout each T_{cycle}. $E_{\text{battery}} = C_{\text{battery}} \cdot V_{\text{nom}}$ represents the energy stored in the node's battery, where C_{battery} signifies the battery's charge in ampere-hours (Ah), and V_{nom} corresponds to the battery voltage (V). For instance, a LoRa node might be powered by a pair of AA batteries with a C_{battery} of 3,000 mAh and a V_{nom} of 1.5V.

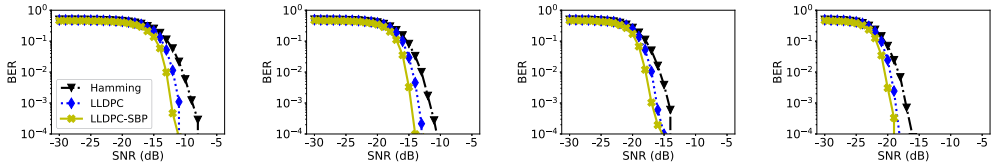
Both T_{cycle} and E_{battery} remain constant. We proceed to determine E_{cycle}. The primary sources of energy consumption are the **microcontroller** (**MCU**) and the Radio. The MCU operates in two modes: active and sleep, while the radio functions in three modes: TX, RX, and sleep. E_{cycle} can be acquired using Equation (11), as described in [51].

$$E_{\text{cycle}} = \left[\left(T_{\text{cycle}} - T_p\right) \cdot \left(P_{M\_off} + P_{R\_off}\right)\right] + \left[T_p \cdot \left(P_{R\_tx} + P_{M\_on}\right)\right]. \tag{11}$$

T_p refers to the packet transmission time within a single sensing cycle. The active mode lasts for T_p, while the sleep mode endures for $(T_{\text{cycle}} - T_p)$. $P_{M_o ff}$ and $P_{R_o ff}$ denote the power consumption of the MCU and Radio in sleep mode, respectively. $P_{M_o n}$ and $P_{R_{tx}}$ indicate the power consumption of the MCU and Radio in TX mode, respectively. The power consumption of the MCU and Radio in these two states is provided in [51].

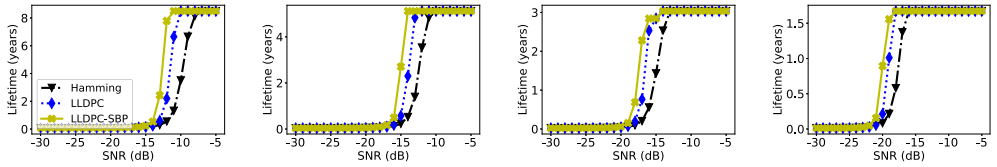
We then need to determine the encoding and transmission duration. We presume the existence of an optimal **Hybrid Automatic Repeat Request** (**HARQ**) mechanism, which retransmits supplementary bits to rectify incorrect bits acquired after FEC decoding. The supplementary transmitted bits' length is twice that of the erroneous bits. *LLDPC* yields a lower post-decoding BER, enabling more reliable communication with significantly fewer re-transmitted bits.

We do not account for the energy consumption associated with awaiting ACKs (in the RX state). According to the LoRaWAN specification, LoRa nodes must receive an ACK within the first or second reception window before transmitting the subsequent packet. These windows are activated



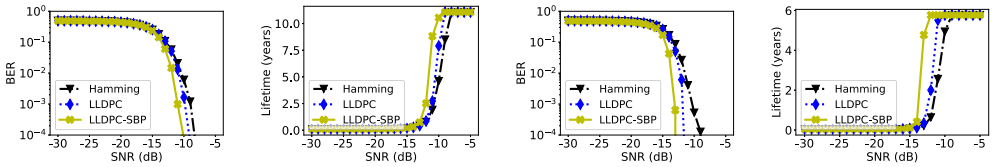
(a) BER *vs.* SNR, SF7. (b) BER *vs.* SNR, SF8. (c) BER *vs.* SNR, SF9. (d) BER *vs.* SNR, SF10.

Fig. 14. The BER curves of three decoding methods for different SFs (CR: 4/6).



(a) Lifetime *vs.* SNR, SF7. (b) Lifetime *vs.* SNR, SF8. (c) Lifetime *vs.* SNR, SF9. (d) Lifetime *vs.* SNR, SF10.

Fig. 15. The lifetime curves of three decoding methods for different SFs (CR: 4/6).



(a) BER *vs.* SNR, CR4/5. (b) Lifetime *vs.* SNR, CR4/5. (c) BER *vs.* SNR, CR4/7. (d) Lifetime *vs.* SNR, CR4/7.

Fig. 16. The BER and lifetime curves of three decoding methods for different CRs when SF7.

at 1 s and 2 s delays following the conclusion of the uplink. Hence, regardless of whether *LLDPC* or Hamming codes are employed, the LoRa nodes' MCU must wait for the same duration with equivalent power consumption. Both systems expend the same energy while awaiting ACKs.

6.1.2 Benchmarks. We compare the performance of *LLDPC* with the following two baselines.

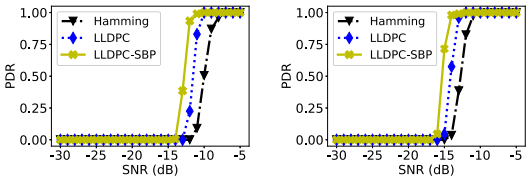
Hamming [25]. Hamming codes serve as the standard FEC codes for LoRa. LoRa employs k/n Hamming codes with $k = 4$ and $n \in 5, 6, 7, 8$, where k signifies the data word length and n represents the codeword length. Hamming codes have a restricted capacity for detecting and rectifying erroneous bits [12, 52]. The 4/5 and 4/6 CRs can solely identify one bit error, while the 4/7 and 4/8 CRs can amend a single incorrect bit per codeword.

LLDPC-SBP. We also execute the SBP algorithm for LDPC decoding, as introduced in Section 2.3. We set the maximum iteration count of *LLDPC*-SBP at 10,000.

6.2 Overall Performance

We assess the efficacy of *LLDPC* under various LoRa configurations, encompassing 4 SFs (i.e., SF7, SF8, SF9, and SF10) and 3 CRs (i.e., 4/5, 4/6, and 4/7). The SNR threshold is defined as the SNR corresponding to a BER of $1e^{-4}$ on the BER-SNR curve, as this BER can attain a PDR of 99.68% with a 40-byte packet size. The outcomes are presented in Figures 14–16.

BER. Figure 14 illustrates the BER performance of distinct SFs for SNR values within the range of $[-30, -5]$ dB, employing a CR of 4/6. *LLDPC* consistently achieves a lower BER than Hamming



(a) PDR *vs.* SNR, SF7. (b) PDR *vs.* SNR, SF8.

Fig. 17. The PDR performance evaluation under two SFs when CR = 4/6.

codes from SF7 to SF10 for all SNR levels. The SNR threshold of *LLDPC* at a given SF nearly matches that of Hamming codes at a higher SF. For instance, by comparing Figure 14(a) with Figure 14(b), the SNR threshold of *LLDPC* at SF7 approaches that of Hamming codes at SF8. In particular, *LLDPC* reduces the SNR threshold by 2.6, 2.3, 2.3, and 2.2 dB from SF7 to SF10 relative to Hamming codes. *LLDPC*-SBP exhibits the lowest BERs, attributable to the SBP algorithm's employment for LDPC decoding, which involves numerous iterations to rectify incorrect bits. However, this comes at the expense of increased decoding latency. *LLDPC* balances BER and decoding latency by implementing the GNN-based BP algorithm. If applications can accommodate a more lenient decoding latency requirement, such as when ACKs are unnecessary, our system can switch to *LLDPC*-SBP for improved BER performance.

Lifetime. We also examine the lifetime of *LLDPC* across diverse LoRa settings and SNR levels, as depicted in Figure 15. *LLDPC* demonstrates a superior lifetime compared to Hamming codes, yielding a consistent lifetime gain. Relative to Hamming codes, *LLDPC* prolongs the median battery life by 51.2%, 31.2%, 13.9%, and 21.2% from SF7 to SF10. Moreover, Figure 15(d) reveals that SF10 offers a lifetime of less than two years, considerably shorter than SF7, as SF10 employs longer symbol duration than SF7.

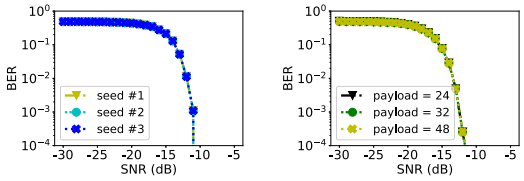
CR. We alter the CRs to scrutinize the impact of distinct CRs on the performance of *LLDPC*. Figure 16 demonstrates that with increasing CR, the SNR threshold decreases, signifying an enhancement in error correction capability. Figure 16(c) unveils that the SNR threshold for Hamming codes with a CR of 4/7 also experiences a reduction compared to CRs of 4/5 and 4/6. This occurs because Hamming codes with a CR of 4/7 can rectify one erroneous bit for every 7-bit codeword.

Packet Delivery Ratio (PDR). We further explore PDR for various SFs at SNR levels spanning $[-30, -5]$ dB using a CR of 4/6. For a specific SNR, the PDR is calculated as the proportion of accurately decoded packets to the total number of packets transmitted at that SNR. As observed in Figure 17, irrespective of the SFs or SNRs, *LLDPC* consistently delivers the highest PDR, corroborating the findings derived from the BER-SNR curves in Figure 14.

6.3 Performance Under Different Settings

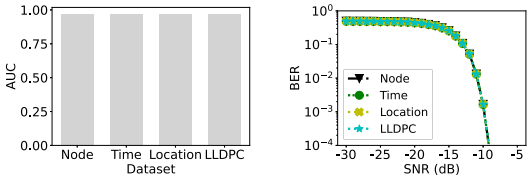
The aforementioned experimental results substantiate the efficacy of *LLDPC*. Then, we examine whether the performance of *LLDPC* is influenced by the parity check matrix and payload length.

6.3.1 Parity Check Matrix. Employing distinct seeds allows us to obtain varying parity check matrices for a specific CR, signifying the altered positions of 1 in rows and columns. We modify the parity check matrix to investigate the potential impact of different matrices on *LLDPC*'s performance. We randomly generate three parity check matrices utilizing three seeds, with SF set to 7 and CR at 4/6. It is important to note that retraining GNN models is necessary for different matrices. Figure 18(a) illustrates that all seeds produce identical BER-SNR curves. This confirms that parity check matrices generated from various seeds do not alter *LLDPC*'s performance.



(a) Parity check matrices. (b) Payload length.

Fig. 18. The performance of *LLDPC* under different settings.



(a) SVM model. (b) GNN model.

Fig. 19. The cross-domain transfer ability analysis.

6.3.2 Payload Length. Considering that distinct payload sizes yield parity check matrices with differing dimensions, we adjust the payload size to evaluate *LLDPC*'s performance. We modify the payload size to 24, 32, and 48 bytes. The SF is set to 8, and the CR is 4/5. The corresponding dimensions of the parity check matrix are 48×240, 64×320, and 96×480, respectively. Retraining GNN models is required for different payload sizes due to the changes in the parity check matrix. Figure 18(b) demonstrates that payload size has no impact on the BER-SNR curves.

6.4 Scalability of *LLDPC*

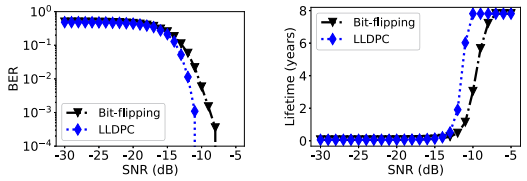
The SVM and GNN models employed in *LLDPC* necessitate prior training, and we aim at assessing the sensitivity of these models to the training data.

We partition the dataset described in Section 4 into multiple subsets based on distinct scenarios, such as LoRa node, location, and time. One subset is utilized to generate synthetic training data, while the remaining subsets are used to evaluate the performance of our SVM and GNN models. For instance, we derive a synthetic subset at 11:00 AM from node A in room R1 for training the SVM and GNN models, and subsequently test the trained model on three separate datasets. The first dataset comprises data collected at 11:00 AM from Node B in Room R1 (denoted as "Node"), the second dataset includes packets sent from Node A in Room R1 at varying times (labeled as "Time"), and the third dataset is obtained from Node A in Room R2 at 11:00 AM (designated as "Location").

Figure 19 presents the results, indicating that the SVM model achieves consistent AUC (Area under the Receiver Operating Characteristic Curve) across the three subsets. The BER-SNR curves of *LLDPC*, when utilizing the other three datasets, exhibit overlapping patterns. This outcome is attributable to our models being trained with a diverse range of SNRs in a fine-grained manner. As SNR is employed to quantify channel quality, our SVM and GNN models remain unaffected by variations in node, time, and location.

6.5 Performance Analysis of *LLDPC* Components

In this section, we assess the efficacy of the two components proposed in *LLDPC*.



(a) BER. (b) Lifetime.

Fig. 20. Performance comparisons between *LLDPC* and bit-flipping algorithm (SF8, CR4/6).

Table 3. Confusion Matrix for SVM Model when SF = 7

		Prediction	
		Correct chirp	Incorrect chirp
Label	Correct chirp	95.8%	4.2%
	Incorrect chirp	0.3%	99.7%

Table 4. Confusion Matrix for SVM Model when SF = 10

		Prediction results	
		Correct chirp	Incorrect chirp
Label	Correct chirp	96.9%	5.1%
	Incorrect chirp	1.0%	99.0%

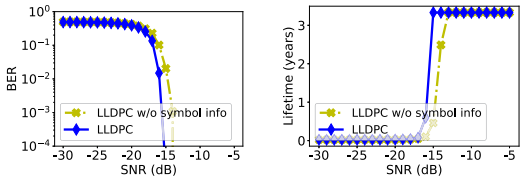
6.5.1 Spectrum-Based LLR Extractor. To validate the performance of the LLR extractor, we examine the disparity between the outputs of LoRa CSS demodulation and our LLR extractor. Specifically, as discussed in Section 2.3, a positive or negative LLR value can be employed to ascertain whether a bit is 1 or 0. If the two methods yield differing values at the same bit position, the discrepancy is incremented by 1. We analyze the differences across millions of packets for various configurations (e.g., SFs and CRs) at distinct SNR levels. Our observations reveal that the difference consistently remains zero, irrespective of the SFs, CRs, or SNRs. This outcome implies that the LLR extractor does not introduce additional erroneous bits but supplies supplementary information, i.e., the confidence of bits.

Figure 20 juxtaposes *LLDPC* with the Bit-flipping algorithm, a hard-decision decoding technique. *LLDPC* can achieve BER and lifetime improvements of 32.1% and 29.2%, respectively. This is because LLR is capable of providing more information than a deterministic 1 or 0.

6.5.2 Symbol-Aware LLR Enhancement. Initially, we present confusion matrices to substantiate the efficacy of our proposed symbol features and SVM models. Subsequently, to determine the performance gain of symbol-aware LLR enhancement, we conduct LDPC decoding with and without this enhancement using a trained GNN model.

Results. Tables 3 and 4 display confusion matrices for SF7 and SF10 at various SNR levels. We observe that the false positive ratio is low, i.e., 0.3%, while the false negative ratio is higher than the false positive ratio. This occurs because incorrect symbols possess distinct features from the correct ones, but some correct symbols exhibit features similar to those of incorrect ones.

Next, we employ the LLR without and with symbol-level information to execute LDPC decoding with trained GNN models. As illustrated in Figure 21, the enhanced LLR can reduce the median BER



(a) BER. (b) Lifetime.

Fig. 21. *LLDPC* with and without symbol-level information when SF8 and CR4/7.

Table 5. Effect of the Length of the Symbol Feature $\mathcal{S}$

$\mathcal{S}$	2	3	5	7	10	20	30
AUC	0.71	0.83	0.97	0.95	0.93	0.90	0.91

by 27.3% at the SNR range of $[-30, -5]$ dB. Thus, it increases the median lifetime by 47.7%. These findings further demonstrate the effectiveness of our symbol-aware LLR enhancement module.

6.6 Parameter Settings

We examine the dimension of the symbol feature $\mathcal{S}$ in *LLDPC*, utilized for SVM classification in Section 3.3. Table 5 presents the AUC of SVM models when varying the dimension of the symbol feature from 1 to 30. We observe that the AUC achieves its maximum value at $\mathcal{S} = 5$. When $\mathcal{S}$ is equal to 2 or 3, the AUC is low, as the feature dimension is insufficient to provide adequate information for symbol classification. However, the feature may introduce noise when the dimension is large (i.e., greater than 5). Therefore, a dimension of $\mathcal{S} = 5$ is recommended for practical applications.

6.7 Campus-Scale Testbed Experiments

In-Field Experiment Design. The synthetic dataset employed in the previous evaluation section contains only additive white Gaussian noise. However, real packets are subject to other types of noise, such as multipath interference. Furthermore, since LoRa utilizes an unlicensed spectrum, co-location of multiple networks sharing the same sub-GHz unlicensed band is inevitable, resulting in cross-technology interference. To assess *LLDPC* with real datasets, we deploy LoRa nodes in various locations, encompassing diverse land cover types (e.g., ponds, trees, and buildings).

We adjust the SFs and CRs according to the link quality. During the real data collection, we also record the SNRs of the received packets to reflect the actual link quality. Consequently, we can simulate LoRa links with SNR variations based on the recorded SNRs. The link adaptation is performed on this simulated link.

Testbed. For the in-field experiments, we deploy LoRa nodes at ten locations, as illustrated in Figure 22. Location #0 is the closest, and location #8 is the farthest. The SF settings at the ten locations are as follows: SF7 is selected at locations #0, #1, and #2; SF8 at locations #3 and #4; SF9 at locations #5 and #9; and SF10 at locations #6, #7, and #8. Each LoRa node transmits 150 packets with a 10-second interval. The payload size is 32 bytes, and the CR is 4/5, resulting in a packet size of 40 bytes. Nodes are equipped with a 3,000 mAh power bank.

6.7.1 Performance on the Real Dataset. We calculate BER and lifetime for each LoRa node.

Results. Figure 23 presents the BER and lifetime of Hamming codes, *LLDPC*, and *LLDPC*-SBP. On average, *LLDPC* significantly outperforms the Hamming codes in terms of BER and lifetime.

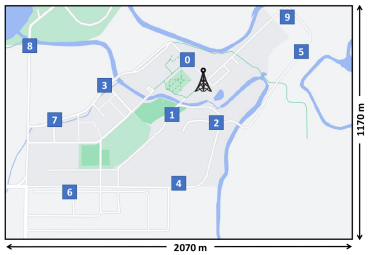
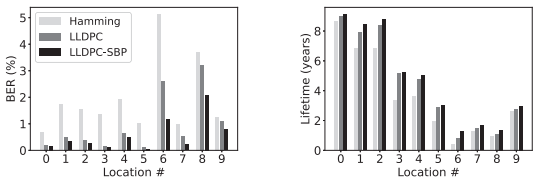


Fig. 22. The illustration of our in-field testbed and the topology of the LoRa nodes and the receiver.



(a) BER. (b) Lifetime.

Fig. 23. BER and lifetime performance at ten different locations on a campus-scale testbed.

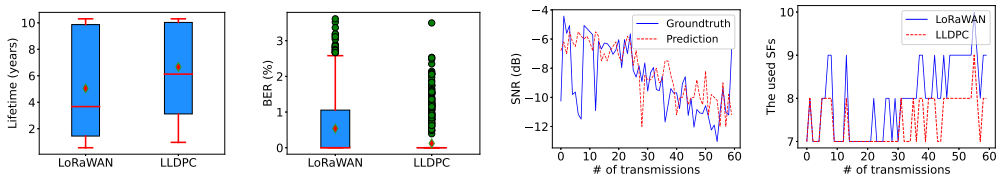
Specifically, *LLDPC* can reduce the average BER of Hamming codes across ten nodes by 50.8% and extend the average lifetime by 19.3%.

In Figure 23(a), *LLDPC* exhibits a lower BER than Hamming codes at all locations. For instance, at location #6, *LLDPC* reduces the BER by 54.5% to 2.63%, while *LLDPC*-SBP further decreases the BER by 48.8% to 1.19%. Despite its proximity to the receiver, location #6 has a higher BER with Hamming codes due to low SNR levels caused by trees and buildings obstructing the signal. We also estimate the lifetime of LoRa nodes at each location. Figure 23(b) reveals that *LLDPC* can extend the lifetime from 3.38 to 5.25 years, while the maximum lifetime reaches 9.08 years with *LLDPC*-SBP at location #0. In comparison to Hamming codes, *LLDPC* significantly reduces the BER and extends the lifetime with a CR of 4/5.

As we increase CR (e.g., 4/6), the performance gains of *LLDPC* become more pronounced than those at a CR of 4/5. This is because the 4/6 Hamming codes are unable to correct any erroneous bits and only add overhead. In contrast, *LLDPC* offers a more robust error correction capability.

6.7.2 Integration of LLDPC into LoRaWAN. We employ simulated links to jointly configure SFs and CRs. In particular, we use the GitHub library [53] to generate LoRa packets with specific SF and CR as determined by the mechanism introduced in Section 5. This library is commonly used for generating LoRa packets in current research [45]. Packets are then transmitted over additive white Gaussian noise channels [54] with an SNR obtained from the collected SNRs. We utilize the three most recently received packets to predict the SNR for the next transmission. In practice, packets are sent with a low-duty cycle, e.g., 10 minutes. Since nodes send packets at 10-second intervals, we can receive 60 packets per 10 minutes. Therefore, to implement a 10-minute duty cycle, we use the first, 61st, and 121st packets to predict the SNR of the 181st transmission, and so on. We run LoRaWAN and *LLDPC*, respectively.

Results. Figure 24(a) displays the lifetime of LoRaWAN and *LLDPC*. We observe that *LLDPC* can extend the lifetime of nodes by 86.7% on average, compared to LoRaWAN. The reasons for this are twofold. Figure 24(c) demonstrates that the link quality varies dynamically from −4 dB to −15 dB, leading to inaccurate SNR prediction. Nevertheless, *LLDPC* offers a high error correction



(a) Lifetime. (b) BER. (c) The prediction of SNR. (d) The selected SFs.

Fig. 24. Joint selection of SF and CR for link adaption.

Table 6. Time Consumption (unit: s) of GNN-Based BP, Soft BP, and *LLDPC* to Decode One Packet for Different CRs on PC/Raspberry Pi 3

	CR = 4/5	CR = 4/6	CR = 4/7
Soft BP	0.86/2.51	1.76/3.27	2.95/5.46
GNN-based BP	0.021/0.434	0.026/0.628	0.068/1.502
LLDPC	0.043/0.548	0.054/0.749	0.097/1.871

Note that GPU is disabled while performing LDPC decoding.

capability to handle imprecise SNR prediction. As shown in Figure 24(b), *LLDPC* achieves a BER reduction of 97.1% compared to LoRaWAN. Secondly, Figure 24(d) indicates that for the same link quality, *LLDPC* tends to use smaller SFs than LoRaWAN. In comparison to the standard LoRa PHY, *LLDPC* can achieve lower SNR thresholds with the same SF and CR. Consequently, for a given SNR, *LLDPC* can attain the same BER with smaller SF and CR, significantly extending the node lifetime.

6.8 Storage Overhead and Running Time

In *LLDPC*, it is necessary to consider the overhead of two components, i.e., gateways and nodes.

6.8.1 Gateways. We execute our SVM and GNN models on both a local PC and a single-board computer, Raspberry Pi 3. The evaluation focuses on storage overhead and running time of our models. Running time, which measures the time required to decode one packet, is determined by executing the process thousands of times and calculating the average. Since the gateway is powered by the grid, we do not consider its energy consumption.

Results. The storage overheads of the SVM and GNN models are 15.7, 16.3, and 16.8 MB for CRs of 4/5, 4/6, and 4/7. We can observe that the model size increases with larger CRs. This is attributable to a larger CR resulting in a larger packet size, necessitating a larger model and additional parameters to manage inputs.

In accordance with the LoRaWAN specification, LoRa nodes must receive an ACK within the first or second receiving window before transmitting the next packet transmission. These two windows open with a delay of 1 s and 2 s following the end of the uplink. Considering this constraint, Table 6 further presents the running time of the SBP algorithm, GNN-based BP, and the entire system. Similar to storage overhead, running time increases with CRs. Our GNN-based BP algorithm demonstrates a 58.09× reduction in average decoding latency compared to Soft BP. We also measure the total elapsed time of *LLDPC*, encompassing the spectrum-based LLR extractor, the symbol-aware LLR enhancement module, and the GNN model. With a high-performance computing computer, it takes under 100 ms. On the Raspberry Pi 3, the time consumed is approximately 1.056 s with a deviation of 0.582 s. For CRs of 4/5 and 4/6, the entire running time is less than 1s, allowing the gateway to transmit ACKs within the LoRa node's first receiving window. *LLDPC*

Table 7. Storage and Energy Consumption of *LLDPC* on LoRa Nodes for MCU and Radio, where Energy is Calculated for Each Packet with a 32-Byte Payload and SF is Set to 7

	CR = 4/5	CR = 4/6	CR = 4/7
Storage (KB)	5.0	6.3	11.0
Energy of MCU (mJ)	0.9	1.5	2.4
Energy of Radio (mJ)	480.7	619.8	777.1

requires more than 1s when CR is 4/7, so we send ACKs during the second receiving window with a two-second time limit.

6.8.2 LoRa Nodes. In *LLDPC*, the overhead of nodes stems from two aspects: storing the parity check matrix and generating encoded bits. Distinct parity check matrices are required for different CRs. By incorporating an SD card module on the Arduino board, parity check matrices with CRs of 4/5, 4/6, and 4/7 can be stored on the Arduino board. To generate encoded bits, the parity check matrix stored on SD card module is read row by row, followed by XOR operations to encode the data.

Results. Table 7 presents storage overheads of parity check matrices under varying CRs. It is evident that matrix size increases with higher CRs, as a larger CR entails a greater number of rows.

Energy consumption associated with LDPC encoding and transmission is also depicted in Table 7. As CRs rise, energy consumption experiences an increase. During the encoding process, the MCU is active with a power of 23.48 mW [51], resulting in an energy consumption of 0.9 mJ (23.48mW $\times$ 42.5ms) when CR is set to 4/5. This demonstrates that for a node with a battery capacity of 3,000 mAh and a voltage of 1.5 V, the encoding process accounts for merely 0.6E-5% of the energy. Such energy consumption is virtually negligible compared to that expended during the transmission process.

7 RELATED WORK

Error Correction in LoRa Networks. Error correction is extensively employed in wireless networks [10, 55–57]. In WiFi and cellular networks, multiple antennas serve to enhance the SNRs of received signals. Recent LoRa studies [12, 49, 50, 58–60] have adopted this concept. Choir [58] improves the SNRs of received signals by deploying multiple co-located LoRa nodes. Charm [49] decodes weak signals by coordinating multiple gateways to detect combined energy peaks in the spectrum. OPR [12] employs disjoint link layer bit errors received by multiple gateways to identify and correct erroneous bits using the CRC defined at the MAC layer. Nephalai [59] conveys compressed PHY samples to the cloud and demodulates these samples using sparse approximation. Furthermore, Chime [50] attempts to circumvent multipath interference by selecting the operating frequency of LoRa nodes, thereby achieving additional SNR gain for LoRa transmissions. Each of these systems necessitates multiple pairs of transceivers. In contrast, *LLDPC* attains supplementary SNR gain via the proposed GNN-based BP algorithm for LDPC decoding, utilizing only a single pair of transceivers. Consequently, *LLDPC* can complement existing work and benefit from the diversity gain provided by multiple gateways.

Elshabrawy et al.[61] theoretically analyze the advantages of employing Non-Binary Single Parity-Check Codes. However, the encoding process of these codes demands arithmetic calculations in GF, which is computationally expensive[16]. In contrast, *LLDPC* employs binary LDPC codes, which involve only simple XOR operations. Such operations can be executed at the sensor node with minimal energy consumption. Additionally, we introduce a novel encoding scheme and design an architecture based on LDPC codes, along with its implementation.

LoRa Throughput. FTrack [62] and CurvingLoRa [63] introduce various approaches to address packet collisions in LoRa [64], consequently enhancing the network's throughput [64, 65]. *LLDPC* focuses on incorporating LDPC codes into LoRa to extend the communication range and optimize energy efficiency, aligning with research in this area.

Deep Learning-based Wireless Communication. Deep learning-based techniques [66–69] have been applied to conventional wireless communication systems, such as error correction codes [70, 71]. Specifically, deep learning-based models [72–74] learn the encoded structure of signals and decode data bits, including convolutional codes [70] and LDPC codes [71]. However, existing deep learning-based approaches are unsuitable for LoRa since they necessitate signals to be above the noise floor. *LLDPC* employs the amplitude spectrum of symbols obtained from LoRa CSS demodulation to extract LLRs, enabling it to combat various types of noise and achieve ultra-low SNR error correction below the noise floor.

For LoRa communication, only a few recent studies have explored deep learning-based approaches [75, 76]. DeepSense [75] investigates deep learning-enhanced random access in the coexistence of LPWANs, even below the noise floor (e.g.,–10 dB). DeepLoRa [76] devises a land-cover-aware path loss model employing bidirectional long short-term memory networks, which reduces link estimation error by 2× compared to the state-of-the-art. We initially demonstrate that a GNN-based LDPC decoder can attain lower SNR thresholds than Hamming codes with manageable computation.

NELoRa [45] introduces a neural-enhanced LoRa demodulation method that leverages the feature abstraction capability of deep learning to facilitate LoRa demodulation under ultra-low SNR. *LLDPC*, on the other hand, focuses on a distinct task, i.e., FEC coding. Both are essential components of the LoRa physical layer. *LLDPC* extracts LLRs from the amplitude spectrum of CSS modulation using the SoftMax operation (Equation (5)). Our LLR extractor can also utilize the output of NELoRa to obtain bit LLRs. This is because NELoRa can generate probabilities for all frequency bins based on its final SoftMax layer. Thus, NELoRa and *LLDPC* can collaborate for more robust communication.

GNN-based BP Algorithm. Deep learning-based methods have been proposed for a range of applications, including wireless networking [45] and smartphone app usage prediction [66, 67]. The belief propagation algorithm involves iterative message passing between bit nodes and factor nodes on factor graphs. Researchers have been exploring the implementation of message exchange using GNN models [41, 77]. NEBP [77] develops a GNN model that operates on the factor graph and exchanges information with the traditional BP algorithm for error correction decoding. However, this approach remains time-consuming and is therefore unsuitable for LoRaWAN. FGNN [41] extends GNNs to factor GNNs, enabling the network to capture higher-order dependencies between bit nodes and factor nodes. Sebastian et al.[78] develop a GNN-based architecture for learning a generalized message-passing algorithm tailored to the FEC code structure. Differing from these approaches, *LLDPC* employs large-scale synthetic LoRa packets to generate the features necessary for the GNN model. This unique method focuses on the specific requirements of LoRa communications. Additionally, *LLDPC* distinguishes itself by empirically selecting the appropriate number of GNN layers, a critical factor in achieving rapid LDPC decoding. This strategic layer selection, combined with the specialized use of synthetic LoRa packets, sets *LLDPC* apart by enhancing its decoding efficiency and effectiveness, specifically tailored for LoRa communication scenarios.

8 DISCUSSION

This section investigates the practicability of deploying *LLDPC* on **Commercial Off-The-Shelf** (**COTS**) hardware, detailing the specific modifications or enhancements necessary to support *LLDPC*.

LoRa Nodes: It is determined that no modifications are required for LoRa nodes to accommodate the implementation of LLDPC. The rationale is that the incorporation of LLDPC necessitates additional memory to archive the predefined parity check matrices for various CRs. Table 7 demonstrates that the maximal memory requisite is approximately 11.0 KB, which is well within the 32 KB Flash Memory capacity of the Arduino Uno Rev3's ATmega328P Microcontroller.

LoRa Gateways: Conversely, LoRa gateways need hardware upgrades to support *LLDPC*. The crux of these upgrades focuses on augmenting gateways to enable the extraction of LLRs and to implement a GNN-based belief propagation algorithm. These upgrades require improvements to the demodulation and decoding modules: *(1) Demodulation:* The modification from merely outputting integer values to providing LLRs for bits is required. *(2) Decoding:* The decoding module must be enhanced to equip the gateway with the capacity to archive predefined parity check matrices and well-trained GNN models for various CRs. Following the procedures of Gray demapping, deinterleaving, and dewhitening, the original bits are reconstructed through LDPC decoding, employing our GNN-based belief propagation algorithm.

9 CONCLUSION

We introduce *LLDPC*, an efficient efficient error correction coding scheme for LoRa networks. *LLDPC* integrates LDPC codes into LoRa by employing the spectrum-based LLR extractor, a symbol-aware LLR enhancement module, and a GNN-based belief propagation algorithm. *LLDPC* demonstrates exceptional error correction capabilities.

REFERENCES

[1] Kang Yang and Wan Du. 2022. LLDPC: A low-density parity-check coding scheme for LoRa networks. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys'22)*.

[2] Jothi Prasanna Shanmuga Sundaram, Wan Du, and Zhiwei Zhao. 2019. A survey on LoRa networking: Research problems, current solutions, and open issues. *IEEE Communications Surveys and Tutorials* 22, 1 (2019), 371–388.

[3] Kang Yang, Yuning Chen, Xuanren Chen, and Wan Du. 2023. Link quality modeling for LoRa networks in orchards. In *Proceedings of the 22nd ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN'23)*.

[4] Kang Yang, Yuning Chen, and Wan Du. 2024. OrchLoc: In-orchard localization via a single LoRa gateway and generative diffusion model-based fingerprinting. In *Proceedings of the 22nd ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'24)*.

[5] Xianzhong Ding and Wan Du. 2022. DRLIC: Deep reinforcement learning for irrigation control. In *Proceedings of the 21st ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN'22)*.

[6] Zehua Sun, Huanqi Yang, Kai Liu, Zhimeng Yin, Zhenjiang Li, and Weitao Xu. 2022. Recent advances in LoRa: A comprehensive survey. *ACM Transactions on Sensor Networks* 18, 4 (2022), 1–44.

[7] Qianyi Huang, Zhiqing Luo, Jin Zhang, Wei Wang, and Qian Zhang. 2022. LoRadar: Enabling concurrent radar sensing and LoRa communication. *IEEE Transactions on Mobile Computing* 21, 6 (2022), 2045–2057.

[8] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. 2017. A survey on LPWA technology: LoRa and NB-IoT. *Ict Express* 3, 1 (2017), 14–21.

[9] Dong-Woo Lim and Kyu-Min Kang. 2023. Robust LoRa modulation scheme in the presence of residual carrier frequency offset. *IEEE Internet of Things Journal* 10, 16 (2023), 14910–14911.

[10] Wan Du, Jansen Christian Liando, Huanle Zhang, and Mo Li. 2015. When pipelines meet fountain: Fast data dissemination in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys'15)*.

[11] Ian F. Akyildiz, Tommaso Melodia, and Kaushik R. Chowdury. 2007. Wireless multimedia sensor networks: A survey. *IEEE Wireless Communications* 14, 6 (2007), 32–39.

[12] Artur Balanuta, Nuno Pereira, Swarun Kumar, and Anthony Rowe. 2020. A cloud-optimized link layer for low-power wide-area networks. In *Proceedings of the 18th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'20)*.

[13] 2013. SX1272/3/6/7/8: LoRa Modem Design Guide. Retrieved July 2013 from https://www.openhacks.com/uploadsproductos/loradesignguide_std.pdf

[14] Wei Song, Yifei Shen, Liping Li, Kai Niu, and Chuan Zhang. 2020. A general construction and encoder implementation of polar codes. *IEEE Transactions on Very Large Scale Integration Systems* 28, 7 (2020), 1690–1702.

[15] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. 1993. Near shannon limit error-correcting coding and decoding: Turbo-codes. In *Proceedings of the 1993 IEEE International Conference on Communications (ICC'93)*.
[16] Stephen B. Wicker and Vijay K. Bhargava. 1999. *Reed-Solomon Codes and Their Applications*. John Wiley and Sons.
[17] Robert Gallager. 1962. Low-density parity-check codes. *IRE Transactions on Information Theory* 8, 1 (1962), 21–28.
[18] Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* 5, 1 (2001), 3–55.
[19] 2018. 5G; NR; Multiplexing and channel coding. Retrieved July 2018 from https://www.etsi.org/deliver/etsi_ts/138200_138299/138212/15.02.00_60/ts_138212v150200p.pdf
[20] Alberto Morello and Vittoria Mignone. 2006. DVB-S2: The second generation standard for satellite broad-band services. *Proceedings of the IEEE* 94, 1 (2006), 210–227.
[21] 2021. IEEE standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)* (2021), 1–4379. DOI: https://doi.org/10.1109/IEEESTD.2021.9363693
[22] Andrea Petroni and Mauro Biagi. 2022. Interference mitigation and decoding through gateway diversity in LoRaWAN. *IEEE Transactions on Wireless Communications* 21, 11 (2022), 9068–9081.
[23] Orion Afisiadis, Matthieu Cotting, Andreas Burg, and Alexios Balatsoukas-Stimming. 2019. On the error rate of the LoRa modulation with interference. *IEEE Transactions on Wireless Communications* 19, 2 (2019), 1292–1304.
[24] Marti A. Hearst, Susan T. Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support vector machines. *IEEE Intelligent Systems and Their Applications* 13, 4 (1998), 18–28.
[25] 2017. LoRaWAN^{TM} 1.1 Specification. Retrieved October 2017 from https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf
[26] Shengguang Hong, Fang Yao, Fengyun Zhang, Yulong Ding, and Shuang-Hua Yang. 2023. Reinforcement learning approach for SF allocation in LoRa network. *IEEE Internet of Things Journal* 10, 20 (2023), 18259–18272.
[27] Ningning Hou, Xianjin Xia, and Yuanqing Zheng. 2023. Jamming of LoRa PHY and countermeasure. *ACM Transactions on Sensor Networks* 19, 4 (2023), 1–27.
[28] Zhenqiang Xu, Shuai Tong, Pengjin Xie, and Jiliang Wang. 2023. From demodulation to decoding: Toward complete LoRa PHY understanding and implementation. *ACM Transactions on Sensor Networks* 18, 4 (2023), 1–27.
[29] David J. C. MacKay. 1999. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory* 45, 2 (1999), 399–431.
[30] R. Tanner. 1981. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory* 27, 5 (1981), 533–547.
[31] 2021. Arduino Uno Rev3. Retrieved from https://store-usa.arduino.cc/products/arduino-un%o-rev3/?selectedStore=us. Accessed Date: 06/19.
[32] Juntan Zhang and Marc P. C. Fossorier. 2004. A modified weighted bit-flipping decoding of low-density parity-check codes. *IEEE Communications Letters* 8, 3 (2004), 165–167.
[33] Jianguang Zhao, Farhad Zarkeshvari, and Amir H. Banihashemi. 2005. On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes. *IEEE Transactions on Communications* 53, 4 (2005), 549–554.
[34] Rolf Johannesson and Kamil Sh Zigangirov. 2015. *Fundamentals of Convolutional Coding*. John Wiley and Sons.
[35] Eugenio Magistretti, Krishna Kant Chintalapudi, Bozidar Radunovic, and Ramachandran Ramjee. 2011. WiFi-Nano: Reclaiming WiFi efficiency through 800 ns slots. In *Proceedings of the 17th ACM Annual International Conference on Mobile Computing and Networking (MobiCom'11)*.
[36] Yong Soo Cho, Jaekwon Kim, Won Y. Yang, and Chung G. Kang. 2010. *MIMO-OFDM Wireless Communications with MATLAB*. John Wiley and Sons.
[37] Srikar Kasi and Kyle Jamieson. 2020. Towards quantum belief propagation for LDPC decoding in wireless networks. In *Proceedings of the 26th ACM Annual International Conference on Mobile Computing and Networking (MobiCom'20)*.
[38] Christopher M. Bishop and Nasser M. Nasrabadi. 2006. *Pattern Recognition and Machine Learning*. Springer.
[39] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008), 1–27.
[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.
[41] Zhen Zhang, Fan Wu, and Wee Sun Lee. 2020. Factor graph neural networks. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS'20)*.
[42] 2020. Semtech SX1276 datasheet. Retrieved May 2020 from https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276
[43] Muhammad Osama Shahid, Millan Philipose, Krishna Chintalapudi, Suman Banerjee, and Bhuvana Krishnaswamy. 2021. Concurrent interference cancellation: Decoding multi-packet collisions in LoRa. In *Proceedings of the 2021 ACM Special Interest Group on Data Communication (SIGCOMM'21)*.

[44] 2021. GR-LoRa. Retrieved 09/17 from https://github.com/rpp0/gr-lora
[45] Chenning Li, Hanqing Guo, Shuai Tong, Xiao Zeng, Zhichao Cao, Mi Zhang, Qiben Yan, Li Xiao, Jiliang Wang, and Yunhao Liu. 2021. NELoRa: Towards ultra-low SNR LoRa communication with neural-enhanced demodulation. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys'21)*.
[46] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.
[47] Davide Magrin, Martina Capuzzo, Andrea Zanella, and Michele Zorzi. 2021. A configurable mathematical model for single-gateway LoRaWAN performance analysis. *IEEE Transactions on Wireless Communications* 21, 7 (2021), 5049–5063.
[48] Rui Fernandes, Miguel Luís, and Susana Sargento. 2021. Large-scale LoRa networks: A mode adaptive protocol. *IEEE Internet of Things Journal* 8, 17 (2021), 13487–13502.
[49] Adwait Dongare, Revathy Narayanan, Akshay Gadre, Anh Luong, Artur Balanuta, Swarun Kumar, Bob Iannucci, and Anthony Rowe. 2018. Charm: Exploiting geographical diversity through coherent combining in low-power wide-area networks. In *Proceedings of the 17th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN'18)*.
[50] Akshay Gadre, Revathy Narayanan, Anh Luong, Anthony Rowe, Bob Iannucci, and Swarun Kumar. 2020. Frequency configuration for low-power wide-area networks in a heartbeat. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*.
[51] Jansen C. Liando, Amalinda Gamage, Agustinus W. Tengourtius, and Mo Li. 2019. Known and unknown facts of LoRa: Experiences from a large-scale measurement study. *ACM Transactions on Sensor Networks* 15, 2 (2019), 1–35.
[52] David J. C. MacKay and David J. C. Mac Kay. 2003. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.
[53] 2022. LoRaPHY. Retrieved January 2022 from https://github.com/jkadbear/LoRaPHY
[54] 2022. AWGN Channel in MATLAB. Retrieved January 2022 from https://www.mathworks.com/help/comm/ug/awgn-channel.html
[55] Aditya Gudipati and Sachin Katti. 2011. Strider: Automatic rate adaptation and collision handling. In *Proceedings of the 2011 ACM Special Interest Group on Data Communication (SIGCOMM'11)*.
[56] Binbin Chen, Ziling Zhou, Yuda Zhao, and Haifeng Yu. 2010. Efficient error estimating coding: Feasibility and applications. In *Proceedings of the 2010 ACM Special Interest Group on Data Communication (SIGCOMM'10)*.
[57] Kang Yang, Miaomiao Liu, and Wan Du. 2024. Rateless-enabled link adaptation for LoRa networking. *IEEE/ACM Transactions on Networking* (2024), 1–16. DOI: https://doi.org/10.1109/TNET.2024.3392342
[58] Rashad Eletreby, Diana Zhang, Swarun Kumar, and Osman Yağan. 2017. Empowering low-power wide area networks in urban settings. In *Proceedings of the 2017 ACM Special Interest Group on Data Communication (SIGCOMM'17)*.
[59] Jun Liu, Weitao Xu, Sanjay Jha, and Wen Hu. 2020. Nephalai: Towards LPWAN C-RAN with physical layer compression. In *Proceedings of the 26th ACM Annual International Conference on Mobile Computing and Networking (MobiCom'20)*.
[60] Zhiwei Zhao, Weifeng Gao, Wan Du, Geyong Min, Wenliang Mao, and Mukesh Singhal. 2023. Towards energy-fairness in LoRa networks. *IEEE Transactions on Mobile Computing* 22, 9 (2023), 5597–5610.
[61] Tallal Elshabrawy and Joerg Robert. 2018. Enhancing LoRa capacity using non-binary single parity check codes. In *Proceedings of the 14th IEEE International Conference on Wireless and Mobile Computing, Networking, and Communications (WiMob'18)*.
[62] Xianjin Xia, Yuanqing Zheng, and Tao Gu. 2020. FTrack: Parallel decoding for LoRa transmissions. *IEEE/ACM Transactions on Networking* 28, 6 (2020), 2573–2586.
[63] Chenning Li, Xiuzhen Guo, Longfei Shangguan, Zhichao Cao, and Kyle Jamieson. 2022. CurvingLoRa to boost LoRa network throughput via concurrent transmission. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*.
[64] Amalinda Gamage, Jansen Liando, Chaojie Gu, Rui Tan, Mo Li, and Olivier Seller. 2023. LMAC: Efficient carrier-sense multiple access for LoRa. *ACM Transactions on Sensor Networks* 19, 2 (2023), 1–27.
[65] Jorge Ortín, Matteo Cesana, and Alessandro Redondi. 2019. Augmenting LoRaWAN performance with listen before talk. *IEEE Transactions on Wireless Communications* 18, 6 (2019), 3113–3128.
[66] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, and Jianhua Zou. 2019. DeepAPP: A deep reinforcement learning framework for mobile application usage prediction. In *Proceedings of the 17th ACM Conference on Embedded Networked Sensor Systems (SenSys'19)*.
[67] Zhihao Shen, Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. 2023. DeepAPP: A deep reinforcement learning framework for mobile application usage prediction. *IEEE Transactions on Mobile Computing* 22, 2 (2023), 824–840.
[68] Miaomiao Liu, Kang Yang, Yanjie Fu, Dapeng Wu, and Wan Du. 2023. Driving maneuver anomaly detection based on deep auto-encoder and geographical partitioning. *ACM Transactions on Sensor Networks* 19, 2 (2023), 1–22.

[69] Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. 2023. ATPP: A mobile app prediction system based on deep marked temporal point processes. *ACM Transactions on Sensor Networks* 19, 3 (2023), 1–24.
[70] Hyeji Kim, Yihan Jiang, Ranvir Rana, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. 2018. Communication algorithms via deep learning. In *6th International Conference on Learning Representations (ICLR)*.
[71] Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J. Gross, David Burshtein, and Yair Be'ery. 2018. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing* 12, 1 (2018), 119–131.
[72] Xianzhong Ding, Wan Du, and Alberto E. Cerpa. 2020. MB2C: Model-based deep reinforcement learning for multi-zone building control. In *Proceedings of the 7th ACM International Conference on Systems for Built Environments (BuildSys'20)*.
[73] Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. 2021. ATPP: A mobile app prediction system based on deep marked temporal point processes. In *Proceedings of the 2021 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS'21)*.
[74] Xianzhong Ding, Wan Du, and Alberto Cerpa. 2019. Octopus: Deep reinforcement learning for holistic smart building control. In *Proceedings of the 6th ACM International Conference on Systems for Built Environments (BuildSys'19)*.
[75] Justin Chan, Anran Wang, Arvind Krishnamurthy, and Shyamnath Gollakota. 2019. DeepSense: Enabling carrier sense in low-power wide area networks using deep learning. *arXiv preprint arXiv:1904.10607* (2019).
[76] Li Liu, Yuguang Yao, Zhichao Cao, and Mi Zhang. 2021. DeepLoRa: Learning accurate path loss model for long distance links in LPWAN. In *Proceedings of the 2021 IEEE Conference on Computer Communications (INFOCOM'21)*.
[77] Victor Garcia Satorras and Max Welling. 2021. Neural enhanced belief propagation on factor graphs. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS'21)*.
[78] Sebastian Cammerer, Jakob Hoydis, Fayçal Aït Aoudia, and Alexander Keller. 2022. Graph neural networks for channel decoding. In *Proceedings of the 2022 IEEE Global Communications Conference Workshops (GC Wkshps'22)*.
[79] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2020), 4–24.

Received 21 December 2023; revised 20 February 2024; accepted 21 May 2024