LLDPC: A Low-Density Parity-Check Coding Scheme for LoRa Networks

Kang Yang
University of California, Merced
Merced, USA

Wan Du
University of California, Merced
Merced, USA

ABSTRACT

Low-density parity-check (LDPC) codes have been widely used for Forward Error Correction (FEC) in wireless networks because they can approach the capacity of wireless links with lightweight encoding complexity. Although LoRa networks have been developed for many applications, they still adopt simple FEC codes, *i.e.*, Hamming codes, which provide limited FEC capacity, causing unreliable data transmissions and high energy consumption of LoRa nodes. To close this gap, this paper develops ***LLDPC***, which realizes **LDPC** coding in **L**oRa networks. Three challenges are addressed. 1) LoRa employs Chirp Spread Spectrum (CSS) modulation, which only provides hard demodulation results without soft information. However, LDPC requires the Log-Likelihood Ratio (LLR) of each received bit for decoding. We develop an LLR extractor for LoRa CSS. 2) Some erroneous bits may have high LLRs (*i.e.*, wrongly confident in their correctness), significantly affecting the LDPC decoding efficiency. We use symbol-level information to fine-tune the LLRs of some bits to improve the LDPC decoding efficiency. 3) Soft Belief Propagation (SBP) is typically used as the LDPC decoding algorithm. It involves heavy iterative computation, resulting in a long decoding latency, which prevents the gateway from sending timely an acknowledgment. We take advantage of recent advances in graph neural networks for fast belief propagation in LDPC decoding. Extensive simulations on a large-scale synthetic dataset and in-filed experiments reveal that *LLDPC* can extend the lifetime of the default LoRa by 86.7% and reduce the decoding latency of the SBP algorithm by 58.09×.

CCS CONCEPTS

• **Networks** → **Network protocol design**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Wireless Systems, Low-Power Wide-Area Networks, LoRa, Forward Error Correction

ACM Reference Format:
Kang Yang and Wan Du. 2022. *LLDPC*: A Low-Density Parity-Check Coding Scheme for LoRa Networks. In *The 20th ACM Conference on Embedded Networked Sensor Systems (SenSys '22), November 6–9, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3560905.3568547

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11…$15.00
https://doi.org/10.1145/3560905.3568547

1 INTRODUCTION

As a promising Low-Power Wide-Area Network (LPWAN), Long Range (LoRa) [1] has been used in many Internet of Things (IoT) applications *e.g.*, smart industry and precision agriculture [2]. LoRa gateways collect data from end devices several miles away at low data rates in unlicensed sub-GHz bands. Interference and ambient noise along with long-distance wireless links degrade the signal-to-noise ratio (SNR) of LoRa packets, causing erroneous bits in the packets received at gateways [3].

Forward Error Correction (FEC) coding corrects bit errors by adding some parity-check bits before each transmission. LoRa adopts Hamming coding, which is simple to implement. However, it is known to be significantly sub-optimal in error correction capacity [3]. SemTech LoRa Design Guide also shows marginal coding gain of Hamming codes under additive white Gaussian noise [4].

Various FEC codes, *e.g.*, Reed-Solomon (RS) codes, Polar codes, and LDPC codes, have been widely used in modern wireless networks [5, 6]. Polar codes require specific hardware circuits to encode data on the sender side [7]. The encoding of RS codes consumes too much energy on sensor nodes as it requires the arithmetic calculation in the Galois Field (GF), which is computation-expensive and memory-intensive [8]. Therefore, they are not suitable for LoRa networks. Compared to other codes, LDPC codes stand out for their superior error correction capability, even approaching the Shannon rate limit [9, 10]. It has been used in 5G New Radio traffic channels [11], satellite communications [12], and the 802.11 WiFi protocol family [13]. In addition, LDPC encoding involves only simple XOR operations, which can be done on sensor nodes without consuming much energy.

In order to bridge the gap between the low FEC capacity of current LoRa networks and the high coding gain provided by LDPC coding, we design an effective LDPC coding scheme for LoRa networks, named *LLDPC*. We tackle the following three challenges.

Firstly, the LDPC coding gain depends heavily on its decoding algorithm, *i.e.*, Soft Belief Propagation (SBP), which takes the Log-Likelihood Ratio (LLR) of each received bit as input. LLR is a floating number that determines not only the bit value but also the confidence of that value. However, the Chirp Spread Spectrum (CSS) demodulation in LoRa specifications [14] does not provide soft information about the output binary sequence. In *LLDPC*, we develop a novel LLR extractor that uses the amplitude spectrum of a symbol to calculate the LLR of each bit in the symbol where a symbol contains a sequence of bits. The amplitude spectrum of a symbol is obtained by CSS demodulation. Specifically, it calculates the LLR of a bit by comparing the amplitudes of all frequency bins in the amplitude spectrum where the bit is 1 or 0.

The second challenge is the low decoding efficiency caused by the large LLR of some erroneous bits. Due to interference or ambient noise, packets may have erroneous bits after demodulation. These

erroneous bits may have large LLRs, which will cause the SBP algorithm to fail. To solve this problem, we try to decrease the LLR of erroneous bits by utilizing symbol-level information. Our key idea is that while we cannot identify the erroneous bits, it is relatively easy to identify the erroneous symbols. The erroneous symbol must contain erroneous bits. Once an erroneous symbol is detected, we directly assign a low LLR to all its bits. By this way, the LLR of the erroneous bit is greatly reduced, avoiding impacting the LDPC coding gain. Toward this end, we need to answer two questions further. *1) how to identify erroneous symbols?* If a symbol is demodulated incorrectly, its amplitude spectrum contains multiple high-amplitude frequency bins. From the amplitude spectrum, we extract a set of features that may characterize the correctness of the demodulated symbols. We use these features to train a binary Support Vector Machines (SVM) classifier [15]. *2) By lowering the LLR of all bits in an identified erroneous symbol, we also reduce the LLR of some correct bits. Does this side effect impact the performance of LDPC coding?* We find that it does not affect LDPC performance through empirical experiments.

Thirdly, the gateway must reply with an acknowledgment (ACK) to the LoRa node within one or two seconds, according to LoRaWAN (Long Range Wide Area Network) specifications [1]. However, the SBP algorithm requires a large number of iterative updating operations, which leads to a long decoding latency. The parity-check matrix of LDPC codes can be transformed into Tanner graphs, a type of factor graphs. It provides us with the opportunity to perform LDPC decoding with Graph Neural Networks (GNN). GNN models can capture higher-order constraints between bit nodes and check nodes on factor graphs, and parameterize the SBP algorithm [16]. We utilize GNN models to perform fast LDPC decoding on Tanner graphs. We conducted experiments on a large-scale synthetic dataset to determine the optimal number of layers of GNNs. The binary cross-entropy loss is used to train GNN models end-to-end.

Finally, we incorporate the above LDPC coding scheme into data rate adaptation process in LoRa networks. In LoRa, the data rate is determined by Spreading Factor (SF). There are two ways to handle a packet transmission failure, *i.e.*, adding more FEC parity-check bits (changing the Coding Rate) or lowering the data rate (changing SF). To transmit a packet, we must decide which FEC Coding Rate (CR) to use for a given data rate. To do so, we first obtain an SF-CR-SNR table by experiments offline. The table records the SNR thresholds for different SFs and CRs. The SNR threshold for a specific combination of SF and CR is obtained from the corresponding BER-SNR curve with a Bit Error Rate (BER) threshold of $1e^{-4}$. We jointly search for the smallest SF and CR in the SF-CR-SNR table to transmit a packet based on the predicted channel SNR.

We implement *LLDPC* on Universal Software Radio Peripheral (USRP) N210 combined with a back-end host. We evaluate *LLDPC*'s performance on a large-scale synthetic dataset and an in-field testbed. Compared with Hamming codes, experiment results show that *LLDPC* can extend the node lifetime by up to 86.7%. Our GNN-based BP algorithm reduces the average decoding latency of the SBP algorithm by 58.09×.

In summary, this paper makes three major contributions:

- To the best of our knowledge, *LLDPC* is the first work to integrate LDPC codes into LoRa networks. We release *LLDPC*'s code on GitHub [17].
- *LLDPC* adopts a set of novel designs, *i.e.*, an amplitude spectrum-based LLR extractor, a symbol-aware LLR enhancement module, and a GNN-based BP algorithm.
- Extensive in-field experiments and simulations based on a large-scale synthetic dataset reveal that *LLDPC* significantly outperforms the standard Hamming codes in LoRa.

2 BACKGROUND AND MOTIVATION

We first introduce the LoRa physical layer (PHY), followed by the motivating experiments. Then, LDPC codes and LLR in the M-QAM (M-ary Quadrature Amplitude Modulation) are presented.

2.1 LoRa PHY

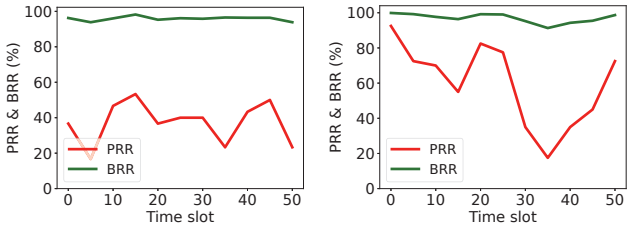
The LoRa PHY of senders mainly performs encoding and modulation. 1) Given a payload (*e.g.*, sensing data), the LoRa PHY first performs a set of encoding operations to improve the over-the-air resilience, including FEC encoding, whitening, diagonal interleaving, and gray mapping. Hamming codes are used as the default FEC coding scheme in LoRa PHY [4]. *LLDPC* will replace Hamming codes as a more effective FEC coding scheme in LoRa PHY. 2) After the above operations, CSS modulates the encoded data into multiple symbols. Each symbol contains a certain number of bits, which is determined by the SF (*i.e.*, 7, 8, 9, and 10). Thus, a symbol can represent an integer in the range of 0 to $2^{SF} - 1$. A symbol is modulated by shifting the initial frequency of a base chirp by a step of $BW/2^{SF}$, where BW is the channel bandwidth. A base chirp is a sinusoidal signal whose frequency linearly increases from 0 Hz to the channel bandwidth, *e.g.*, 125 KHz [4].

At the receiver, LoRa implements demodulation and decoding. Demodulation recognizes a symbol's value by measuring its chirp's initial frequency. It multiplies the received signal by a down-chirp whose frequency decreases linearly over time and performs FFT (Fast Fourier Transform) on the resulting signal. Then, we can obtain the amplitude spectrum of each received symbols. The index of each frequency bin corresponds to a possible symbol value. The value of a symbol is identified by the index of the frequency bin with the highest amplitude. The bits of all the recognized symbols in a packet are concatenated into a binary sequence that is further processed by Gray demapping, deinterleaving, and dewhitening in sequence. Finally, the original bits can be obtained by FEC decoding.

2.2 Motivating Experiments

We investigate the Packet Reception Ratio (PRR) and Bit Reception Ratio (BRR) of LoRa links for two CRs of Hamming codes, i.e., 4/5 and 4/7. For CR of 4/5, every four bits are attached with a one parity-check bit that can detect a one-bit error but cannot correct any error in the five bits. For CR of 4/7, Hamming codes only can correct one erroneous bit for every seven bits. A LoRa node periodically sends packets to the gateway with SF10. The experiment lasts for 2.5 hours. We measure PRR and BRR every three minutes, and the packet size is 32 bytes.

Figure 1 depicts the PRR and the BRR of LoRa links for two CRs. The high BRR and low PRR reveal that the number of erroneous bits in corrupted packets is small, even though many packets are corrupted during transmissions. This phenomenon has also been observed in [3]. Hamming codes cannot correct this small number



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

Figure 1: The Packet Reception Ratio (PRR) and Bit Reception Ratio (BRR) for different CRs of Hamming codes.

of erroneous bits, which inspires us to develop a new FEC coding scheme for LoRa networks, *e.g.*, 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 concatenating M parity-check bits to the K payload bits, where $M = N - K$ and CR is K/N. A sparse parity-check matrix $\mathcal{H}_{M\times N}$ can be generated randomly to obtain an (N, K) LDPC code [9]. The $\mathcal{H}_{M\times N}$ can be represented by the Tanner graph [18], as shown in Figure 2. Nodes labeled with f_i are check nodes, and nodes marked with b_j are bit nodes. Each row in $\mathcal{H}$ represents a check node constraint, *i.e.*, the XOR sum of the participating bit nodes is zero. If $h_{ij} = 1$, it indicates that the bit node b_j is involved in the constraint of the check node f_i. The matrix $\mathcal{H}$ is generated offline. To encode a payload, the sender can easily generate the encoded data by simple XOR operations with the matrix $\mathcal{H}$. Hence, the LDPC encoding process is computationally light and can be implemented on LoRa nodes, *e.g.*, Arduino Uno board [19] in our implementation.

LDPC Decoder. There are two LDPC decoding algorithms, *i.e.*, Bit-flipping [20] and Soft Belief Propagation (SBP) [21]. Bit-flipping is a hard-decision decoding algorithm that takes a binary bit stream as input to decode the data. SBP is a soft-decision decoding algorithm that considers the reliability of received bits by taking LLR as input to form better estimates. Soft-decision decoding performance is better than hard-decision decoding with an average SNR of 2.5 dB [22]. Therefore, this paper is focused on SBP. The SBP decoding algorithm is summarized as follows [21].

- *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 first sends its LLR to its connected check nodes.
- *Step 1: Updating the 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 $\lambda_{b'_j\to f_i} = \text{LLR}\left(b'_j\right)$ for the first iteration.

- *Step 2: Updating the 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 the 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 the 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 value, *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.

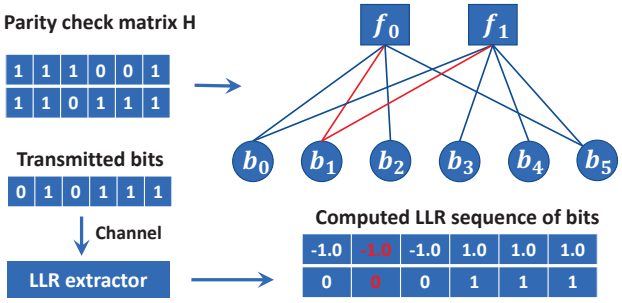


Figure 2: The upper part is an example of a parity-check matrix $\mathcal{H}$ and its corresponding Tanner graph. Each row in $\mathcal{H}$ is one check node (square), and each column denotes one bit node (circle) in the right graph. The lower part is an example of the calculated LLR of six bits.

2.4 LLR in M-QAM

LLR is required for the LDPC decoding. We can compute it by M-QAM demodulation, which is widely adopted in modern wireless networks, such as light communication, WiFi, and 5G [23–26]. It is calculated 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 $\mathbf{r}$ denotes a received symbol and b_j is the j-th bit in the symbol $\mathbf{r}$. When a symbol $\mathbf{r}$ is received, it is represented as a point in the constellation diagram (I-Q plane) where the M standard symbols have fixed positions. Next, the Euclidean distances between the received symbol $\mathbf{r}$ and the M standard symbols in the constellation diagram are computed. Finally, $\mathcal{F}\left(b_j = 1 \mid \mathbf{r}\right)$ is calculated as the sum of the Euclidean distances between the received symbol $\mathbf{r}$ and $M/2$ standard symbols whose j-th bit is 1. Similarly, $\mathcal{F}\left(b_j = 0 \mid \mathbf{r}\right)$ is the sum of the Euclidean distances between symbol $\mathbf{r}$ and the remaining $M/2$ standard symbols whose j-th bit is 0.

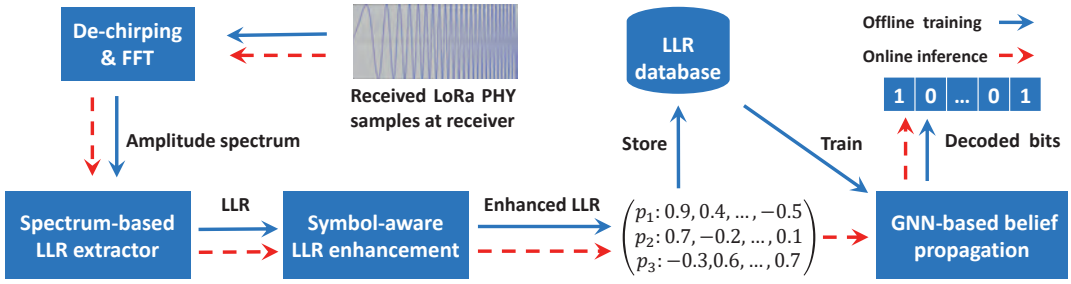


Figure 3: The overall architecture of *LLDPC*.

According to Equation 4, LLR can be used to determine the value of bits, *i.e.*, if $LLR(b_j) \geq 0$, then $b_j = 1$; otherwise $b_j = 0$. In addition, LLR can also provide the confidence that bit b_j is 1 or 0. If the absolute value of the LLR is high, then the confidence level of the bit value is high.

Challenges of Realizing LDPC in LoRa. LoRa adopts CSS modulation, which is different from M-QAM modulation. There are no existing methods to calculate the LLR of the received bits during CSS demodulation. Furthermore, SBP requires many iterations to achieve effective error correction, resulting in long decoding latencies (5.46 seconds in our implementation on a Raspberry Pi 3 single-board computer). However, according to LoRaWAN (Long Range Wide Area Network) specification [1], the gateway must reply with an ACK to the LoRa node within one second. Despite some parallel implementations of LDPC decoding [27], they require high-end hardware or quantum computing platforms, which cannot be used in low-cost, large-scale LoRa networks.

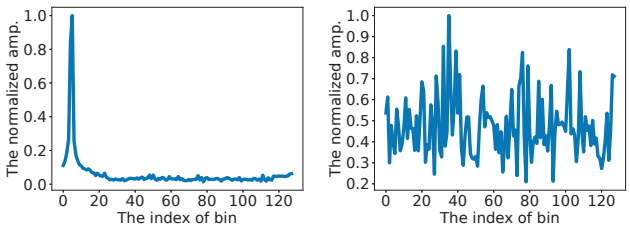
3 DESIGN OF *LLDPC*

In this section, we introduce the design of *LLDPC*, including spectrum-based LLR extractor, symbol-aware LLR enhancement module, and GNN-based belief propagation algorithm.

3.1 Overview

Figure 3 shows the architecture of *LLDPC*. When a LoRa gateway receives a signal from a sensor node, it first obtains the amplitude spectrum of the received symbols through de-chirping and FFT operations. Based on the spectrum results of each symbol, our LLR extractor computes the LLRs of all bits in the symbol (Section 3.2). To enable the best SBP decoding efficiency, we fine-tune the LLR of all bits by leveraging symbol-level information (Section 3.3). Finally, *LLDPC* passes the LLR sequence to the GNN model to perform LDPC decoding (Section 3.4). The decoding latency is low (less than two seconds), which allows the gateway to send back an ACK packet to the sensor node within the time constraint (one or two seconds) specified by the LoRaWAN standard.

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



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

Figure 4: The amplitude spectrum obtained from CSS demodulation for two symbols at different SNRs.

3.2 Spectrum-based LLR Extractor

3.2.1 CSS Demodulation. After performing de-chirping and FFT, we can obtain the amplitude spectrum of a received symbol [14]. The index of frequency bin with the highest amplitude will be selected as that symbol's value. This CSS demodulation process can be treated as a classification task, which outputs probability of all frequency bins. We perform the SoftMax operation [28] on the amplitude spectrum to compute the probability of each bin.

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

where t is the modulated value of a transmitted symbol, exp is the exponential function, and $\mathcal{P}(t = \text{bin}_x)$ denotes the probability that the modulated value equals the index of the x-th frequency bin (*i.e.*, x). The A_{bin_x} is the amplitude of the x-th bin.

The frequency bin with the highest probability is selected as the demodulation result. If the selected probability (*i.e.*, amplitude) is significantly higher than the others, we have higher confidence level in the classification result, as shown in Figure 4(a) for the correctly demodulated symbol. On the contrary, if the highest amplitude in the amplitude spectrum is similar to the amplitude of other frequency bins, it is hard to determine the value of that symbol, as shown in Figure 4(b). However, LoRa CSS demodulation simply selects the index of the frequency bin with the highest amplitude as the symbol's value, ignoring the confidence information.

3.2.2 LLR Calculation. Each symbol contains SF bits. Let $S^+_{b_j}$ be the set of symbols whose j-th bit is 1, and $S^-_{b_j}$ be the set of symbols whose j-th bit is 0 ($0 \leq j < SF$). For example, if SF is 3, the $S^+_{b_0} = \{4,\ 5,\ 6,\ 7\}$ and $S^-_{b_0} = \{0,\ 1,\ 2,\ 3\}$. We compute the LLR of the j-th

bit in a symbol as follows.

$$\mathrm{LLR}\left(b_j\right) = \log \frac{\sum_{t \in S^+_{b_j}} \mathcal{P}\left(t \mid \tilde{r}\right)}{\sum_{t \in S^-_{b_j}} \mathcal{P}\left(t \mid \tilde{r}\right)} \approx \log \frac{\max_{t \in S^+_{b_j}} \mathcal{P}\left(t \mid \tilde{r}\right)}{\max_{t \in S^-_{b_j}} \mathcal{P}\left(t \mid \tilde{r}\right)} \quad (6)$$

where $\mathcal{P}\left(t \mid \tilde{r}\right)$ represents the probability that the value of the transmitted symbol is t given the received signal $\tilde{r}$. It is calculated by Equation 5 using the amplitude spectrum of the symbol. Hence, Equation 6 defines that LLR $\left(b_j\right)$ is the sum of the probabilities that all symbols in set $S^+_{b_j}$ agree on $b_j = 1$, divided by the sum of the probabilities that all symbols in set $S^-_{b_j}$ agree on $b_j = 0$.

We approximate the sum of the probabilities as the highest probability due to two reasons. When SF is large, *e.g.*, SF10, each set ($S^+_{b_j}$ and $S^-_{b_j}$) contains 512 items. It takes time to calculate the probability of each symbol and the sum of all probabilities in each set. Second, the summation may be corrupted by a single outlier, while the maximization operation is more robust. The approximation is also widely used in the existing wireless networks [24]. We also conduct experiments to compute the difference between the sum and max operation. For each packet, we calculate the LLR using sum and max, respectively. Then we 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, implying a negligible effect on the LLR calculation.

The LLR can also be used to determine the bit values. If the LLR of a bit is positive, it is 1; if its LLR is negative, it is 0. The experiments in Section 6 show that our LLR extractor can output exactly the same binary sequence as LoRa CSS demodulation. Therefore, our LLR extractor does not change the BER of the received packets. For example, if the original bit is 1, but the CSS demodulation outputs 0, our LLR extractor will report a negative LLR. Bit errors are caused by interference or ambient noise during wireless transmissions. They will be corrected by FEC codes.

3.3 Symbol-Aware LLR Enhancement

In this section, we first investigate whether the computed LLR of erroneous bits impacts the efficiency of the SBP algorithm. Then, we propose a symbol-aware LLR enhancement module to integrate symbol-level information to fine-tune the LLR of some bits.

3.3.1 The Decoding Efficiency of SBP and the LLR of Erroneous Bits. Figure 2 depicts a simple case where the LLRs of all bits are normalized, and we assume that they all 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 shown in Figure 2. We assume the second bit (b_1) is corrupted, *i.e.*, the second bit is toggled from 1 to 0.

Next, we show how SBP tries to correct the second bit during the decoding iteration. In the first iteration, once bit nodes transmit their initial LLRs to their connected check nodes, SBP updates the

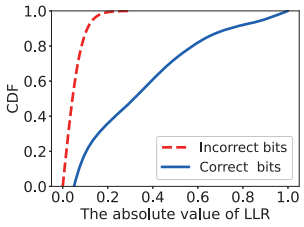


Figure 5: The CDF of LLR for correct and erroneous bits w/o symbol-level info.

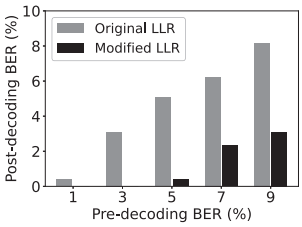


Figure 6: The post-decoding BER by the SBP with original LLRs and enhanced LLRs.

check node message with Equation 7.

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

This equation is instanced from Equation 1. Both messages will be transmitted to bit node b_1. Based on these two messages and its current LLR, bit node b_1 will update its LLR to -0.893 using Equation 3. After this iteration, the LLR of the second bit reduces from -1.0 to -0.893. We run 1,000 iterations. However, this value eventually converges to -0.77, meaning the second bit is still 0. Thus, SBP cannot correct this erroneous bit even with a large number of iterations because of its large absolute LLR value.

Based on this example, we know that the efficiency of the SBP is affected by the large LLR of erroneous bits. We further investigate the severity of the large absolute LLR values for erroneous bits. We plot the CDF of the absolute LLR values for correct and erroneous bits based on a synthetic dataset collected in Section 6. Figure 5 shows that the absolute LLR values of erroneous bits could be up to 0.32. Such a large LLR valve will make SBP less efficient.

Simultaneously, the decoding efficiency of SBP is also investigated when the initial LLR of the second bit is manually changed from -1.0 to -0.1. SBP performs the same updating process. We find that after two iterations, the LLR of the second bit is updated from -0.1 to 0.007, enabling SBP to correct errors successfully. Thus, if we can reduce the absolute LLR value of erroneous bits, we will increase the error correction efficiency of SBP.

We perform a simulation experiment to verify the above observation further. The LLRs of all bits are randomly generated based on their bit values, *e.g.*, if a bit value is 1, its LLR is a positive float number. The erroneous bits are produced with specific pre-decoding BER. We refer to the BER before the LDPC decoding as pre-decoding BER and the BER after the LDPC decoding as post-decoding BER. Figure 6 shows that if the absolute LLR value is comparable with correct bits, SBP cannot correct erroneous bits, although the BER is low (gray bar). However, if we modify the absolute LLR value of the erroneous bits to a small float number, SBP can provide a higher error correction capability (black bar).

However, it is challenging to identify erroneous bits. Fortunately, we can explore some symbol-level information to identify erroneous

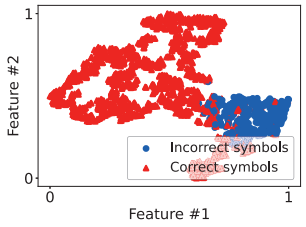


Figure 7: The representation visualization with t-SNE for our proposed symbol features when SF=10.

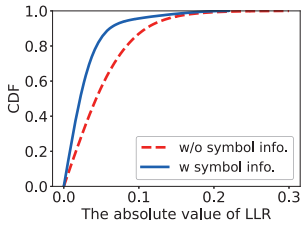


Figure 8: The CDF of the absolute value of LLRs for erroneous bits without and with symbol-level information.

bits indirectly. If a bit is wrong, the symbol containing that bit must be wrongly demodulated. If we can predict that a symbol is demodulated incorrectly, we assign a low LLR to all bits in the symbol. In this way, those erroneous bits will have a lower LLR, which can significantly increase the decoding efficiency of SBP.

3.3.2 Erroneous Symbol Detection. We put forward some symbol features to classify the correctness of symbols by SVM models.

Feature Extraction. To find some compelling features about the correctness of the symbols, we examine the distribution of the amplitude spectrum. Figure 4 shows that if a symbol is demodulated incorrectly, the spectrum contains multiple high-amplitude frequency bins compared to the highest-amplitude bin. We quantify these high-amplitude bins by the amplitude ratio between the highest amplitude and their amplitudes. Specifically, frequency bins are sorted by their amplitudes. Then, we calculate the amplitude ratios of the top-$\mathcal{S}$ high-amplitude bins compared to the highest-amplitude bin. These ratios form a feature vector of length $\mathcal{S}$. We conducted experiments to set $\mathcal{S}$ as five empirically.

To understand the effectiveness of our proposed features, we employ the t-distributed Stochastic Neighbor Embedding (t-SNE) [29, 30] technique to visualize them. From Figure 7, we can find that symbols exhibit a high clustering effect. Therefore, we can use our proposed features to do binary symbol classification, *i.e.*, predicting whether the symbols are correctly demodulated or not.

Classifier. Toward this end, we use the SVM classifier [15] with radial basis function (RBF) kernel to perform symbol classification. In Section 6.5, experiments show that our SVM model can predict the correctness of the symbols with a false negative ratio and a false positive ratio of 4.2% and 0.3%.

3.3.3 The Calculation of the Enhanced LLR. After classifying the correctness of symbols, we enhance the LLR of the bits belonging to the symbols predicted to be incorrectly demodulated. We multiply the probability of a symbol being correctly demodulated by the original LLR of bits. This probability is obtained from the SVM classifier. By this way, we can reduce the absolute LLR value of erroneous bits. Figure 8 shows the CDF of absolute LLR before and after the integration of symbol-level information. We can observe that it does reduce the absolute LLR values of erroneous bits.

On the other hand, we also decrease the absolute LLR values of some correct bits. Does it curb the performance of the SBP algorithm? We conduct a similar simulation with Figure 6. The difference is that we not only reduce the absolute LLR value of erroneous bits but also randomly reduce the absolute LLR value of correct bits. We find that those reduced LLRs of correct bits almost do not harm the SBP decoding performance. The rationale is that lowering LLR for correct bits at least does not negatively impact the SBP updating. However, by assigning a small LLR to erroneous bits, the significant impact of erroneous bits on SBP can be greatly reduced.

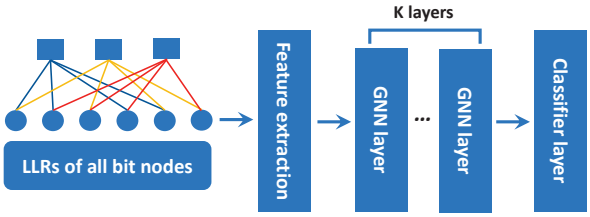


Figure 9: The architecture of the GNN-based BP algorithm.

3.4 GNN-based Belief Propagation

The GNN-based model have been successfully applied to graph-structured data to capture pairwise dependencies between variables and to propagate information throughout the graph [16]. Furthermore, the GNN model is generalized to capture the higher-order constraints between bit nodes and check nodes on factor graphs, and to parameterize the SBP algorithm [16]. Inspired by the above recent advances in GNN, we utilize the GNN model to perform fast LDPC decoding on Tanner graphs.

Figure 9 shows the architecture of the GNN-based belief propagation algorithm. The Tanner graph and enhanced LLRs of each bit are passed to the feature extraction module. The module extracts five matrices that serve as inputs to the GNN layers. These GNN layers can learn high-order constraints among nodes and parameterize the SBP algorithm. Finally, we use a classifier layer to obtain the final bit values.

Feature Extraction. This module generates node features and edge features. The node feature contains two types of features, *i.e.*, the bit node feature and the check node feature. The initial LLR of the bit nodes is used as the bit node feature. For the check node feature, we concatenate the features of all bit nodes connected to the check node as its feature. The edge feature is obtained by concatenating the bit node feature and the check node feature if there is an edge between the bit node and check node. At the same time, we also generate two matrices, *i.e.*, the bit node matrix and the check node matrix. The bit node matrix indicates which check nodes each bit node is connected to, and the check node matrix specifies which bit nodes each check node is linked to. These two matrices represent the structure of the Tanner graph.

GNN Layer. The GNN layer will update the bit node features according to the node features, edge features, and graph structures.

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

where $\mathcal{Q}$ maps edge futures $\mathbf{e}_{p_i \to b_j}$ to a weight matrix, and $\mathcal{Q}$ maps check node feature $\mathbf{f}_{p_i}$ and bit node feature $\mathbf{v}_{b_j}$ to a feature vector. Then an updated bit node feature $\tilde{\mathbf{v}}_{b_j}$ can be generated through matrix multiplication and summation operations.

In this GNN layer, we need to choose the number of GNN layers $\mathcal{K}$. We conduct experiments to empirically set the value of $\mathcal{K}$. We

randomly select 75% of the dataset to train GNN models with different 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, and we measure the decoding latency on a local PC and a single-board computer of Raspberry Pi 3. Table 1 shows the effect of $\mathcal{K}$ on the decoding performance. We can see that the choice of $\mathcal{K}$ significantly impacts decoding latency but less on decoding accuracy. For satisfactory performance and acceptable decoding latency, we set the number of layers $\mathcal{K}$ as 8.

Classifier Layer. Finally, the updated bit node's feature $\tilde{\mathbf{v}}_{b_j}$ will be passed to the classifier layer, which is used to determine the bit value, *i.e.*, 0 or 1.

4 MODEL TRAINING FOR *LLDPC*

We introduce 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, *i.e.*, the SVM model for symbol-aware LLR enhancement module (Section 3.3) and the GNN-based BP algorithm (Section 3.4). *LLDPC* is implemented on a local computer with one CPU that has Intel(R) Core (TM) i9-11900KF @ 3.50 GHz with 16 cores. A graphics processing unit (GPU) card (NVIDIA GEFORCE RTX 3080 Ti) is used to accelerate the training process of GNN modules.

Implementation. LoRa nodes are hand-crafted with SX1276 Radio [31] on the Arduino Uno host boards [19]. We use the USRP N210 platform for capturing over-the-air LoRa signals, operating on a UBX daughter board at the 904.3MHz bands. The sampling rate is 1 MHz. The captured signal samples are then delivered to a back-end host for demodulation [32, 33].

We cannot disable Hamming codes due to hardware limitations. Alternatively, on the sender side, we encode a 32-byte payload data into a 40-byte encoded data by LDPC encoding (CR is 4/5). The 40 bytes of data are then sent to the receiver by modulated symbols after successive Hamming encoding, whitening, diagonal interleaving, and gray mapping in the sender's hardware. Regardless of the CR of LDPC, the CR of Hamming codes is always set to 4/5. For CR of 4/5, every four bits are concatenated with one parity-check bit. For Hamming codes, the CR of 4/5 can detect one erroneous bit but cannot correct any error in the five bits.

At the receiver side, we utilize the amplitude spectrum of the demodulated symbols to calculate the LLR of bits, where bits contain the encoded data from LDPC and Hamming codes. Then, we perform Gray demapping, deinterleaving, and dewhitening in sequence on the calculated LLR. We drop one parity-check bit and extract only the four data bits' LLR for every five bits (Hamming codes' CR is 4/5). The impact of Hamming codes on LDPC decoding performance is eliminated in this way. All these operations are implemented in the public GitHub libraries [32, 33]. Thus, we can get the LLR with 40 bytes of data bits encoded by LDPC codes. A symbol-aware LLR enhancement module enhances these LLRs. We run the GNN-based BP algorithm to perform LDPC decoding.

Training Data Collection. We collect the LoRa I/Q signals with USRP N210 to generate training datasets. LoRa nodes transmit random packets periodically at five locations in an office building. The 12,000 packets are collected at high SNR (>20dB) with different transmission settings, including 4 SFs (*i.e.*, 7, 8, 9, 10) and 3 CRs (*i.e.*, 4/5, 4/6, 4/7). Based on the collected high SNR packets, we use data augmentation to generate 3.6 million packets with SNR from -35 dB to -5 dB in 0.1 dB steps. To generate new LoRa packets with a specific SNR, we add Gaussian white noises with corresponding amplitudes on the collected I/Q samples. It is a widely used data enhancement method [34, 35].

Table 1: The 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, where CR is 4/5 and SF is 7. The latency is measured from two platforms, *i.e.*, PC/Raspberry Pi 3.

# 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

Training SVM model. To train the SVM model, we first extract the feature vectors of symbols on the amplitude spectrum. By conducting experiments, the length of the feature vector is set to 5. The label of a symbol can be obtained by comparing the demodulated value of the symbol with its actual value. The SVM models use the Radial Basis Function (RBF) as the kernel. One parameter must be determined for the RBF kernel, *i.e.*, $\mathcal{C}$. Parameter $\mathcal{C}$ trades off the misclassification of training examples against the simplicity of the decision surface. A low $\mathcal{C}$ makes the decision surface smooth, while a high $\mathcal{C}$ aims at classifying all training examples correctly. $\mathcal{C}$ is empirically set to 1.0. Additionally, considering the unbalanced number of two classes, we set the field of *classWeight="balanced"* to automatically adjust the sample weight inversely proportional to the number of classes.

Training GNN model. The binary cross entropy loss $\mathcal{L}$ for the GNN model is computed 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 by the LoRa receiver during the training stage. After calculating the loss, we back-propagate the loss to the network to update GNN modules.

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

where Θ denotes all the parameters of the GNN-based BP algorithm. We train the GNN model using the enhanced LLRs and the labels of each bit. We construct a GNN model consisting of eight GNN layers. Each layer shares the same Q_1 and Q_2 functions, which are a two-layer Multilayer Perceptron (MLP) network as follows MLP (64) - MLP (4). The first layer comes with a ReLU activation function and the second layer is with no activation function. The model is implemented using PyTorch [36], trained with Adam optimizer [37] with an initial learning rate of 0.01, and after every 10000 samples, the learning rate is decreased by a factor of 0.98. The parameters of the GNN model are uniformly initialized to $[-0.1, 0.1]$. The batch size is set as 32.

5 INTEGRATION OF *LLDPC* INTO LORAWAN

In the above two sections, we have developed our LDPC decoding scheme on the LoRa receiver side. In this section, we further propose a data rate adaptation mechanism to jointly set SFs and CRs of LoRa senders by considering the *LLDPC*-based error correction system.

LoRaWAN specifications [1] use the Adaptive Data Rate (ADR) algorithm to select SF for each sender node. It first estimates the SNR of a link by averaging the SNRs of the recent multiple received packets. It then chooses the highest data rate whose SNR threshold is lower than the estimated SNR.

The candidates for CRs could be 4/5, 4/6, and 4/7. In *LLDPC*, the LDPC codes with different CRs for one SF also can provide different SNR thresholds, as shown in Table 2. Hence, we have to set SFs and CRs jointly. We determine the SNR threshold when BER is $1e^{-4}$ since this BER can achieve a 99.68% of packet delivery rate when the packet size is 40 bytes.

When a packet is received from a LoRa node, the gateway predicts the SNR for the subsequent transmission by weighting the average of the three most recently received packets. Based on the predicted SNR, we query Table 2 to obtain the candidate of SFs and CRs whose SNR threshold is lower than the predicted SNR. The final SF and CR are chosen with the shortest transmission time, which is calculated by the LoRa standard [4]. The updated SF and CR will then be sent back to the LoRa node. The LoRa node will use the most recently received SF and CR to transmit the next packet.

Disabling LDPC Codes. The CR of 4/4 is also included in Table 2. Adopting LDPC codes always carries overhead from parity-check bits. Thus, we should consider the CR of 4/4. If CR is 4/4, LDPC codes will not be used. For example, if the predicted SNR is greater than -7.5 dB, we can use SF7 and a CR of 4/4 because the predicted SNR is larger than the SNR threshold of SF7 and CR of 4/4, which has the shortest transmission time among all the available settings. In this case, LDPC codes are not required.

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 [4].

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

6 EVALUATION

We conduct extensive experiments to evaluate *LLDPC* on large-scale synthetic datasets and in-field experiments.

Evaluation on Synthetic Datasets. The synthetic dataset allows us to manipulate SNRs in a fine-grained manner. It enables us to evaluate the performance of *LLDPC* under comprehensive settings, including all available SFs, CRs, and SNRs. Additionally, it powers us to have enough data to train GNN models. Section 6.2 first presents the overall performance of *LLDPC* on the large-scale synthetic dataset, followed by the performance of *LLDPC* under different experiment settings in Sections 6.3 and 6.4. We also evaluate the effectiveness of our proposed components in Section 6.5.

In-Field Experiments. Since the synthetic dataset contains only artifact Gaussian white noises in the indoor environment, we further test *LLDPC* in the campus environment. Therefore, in-field experiments are used to assess the performance of *LLDPC* when additive white Gaussian noise and other types of noise (such as multipath interference) are present. Section 6.6.1 evaluates the performance of *LLDPC* at each location in in-field experiments. So far, we have evaluated the performance of *LLDPC* under different SFs and CRs separately. Next, we will test the performance of *LLDPC* by integrating it into LoRaWAN. In section 6.6.2, our system selects SFs and CRs to accommodate link quality. Finally, we measure the overhead of *LLDPC* in Section 6.7.

6.1 Experimental Setup

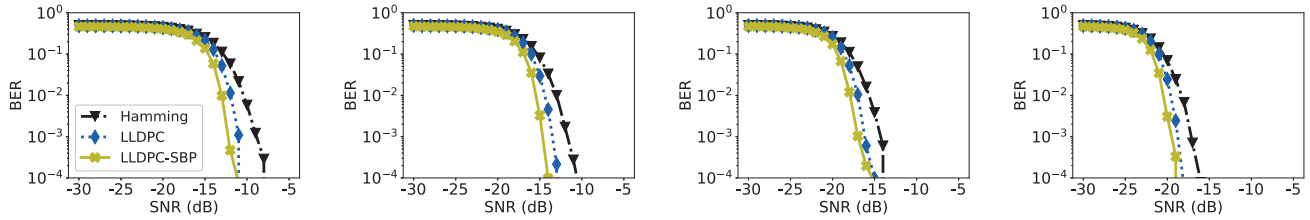
6.1.1 Performance Criteria. We adopt the following two metrics to evaluate the performance of *LLDPC*.

Bit Error Ratio (BER). It is the number of erroneous bits divided by the total number of bits in a packet. We compute the BER once for each transmitted packet.

Lifetime. The lifetime of a node is the duration from the first startup to the exhaustion of its battery energy. This metric measures the energy efficiency of *LLDPC*. It is calculated using LoRaWAN battery models [38–40]. Specifically, the consumed energy consists of two parts, *i.e.*, the energy consumed from the microcontroller (MCU) and the Radio during the encoding and transmission processes. During the encoding process, only the MCU is busy. In the transmission state, both the MCU and the Radio are active. The power consumption of MCU and Radio in two states is provided in [38]. We need to compute the encoding and transmission time. We assume that there is an ideal Hybrid Automatic Repeat Request (HARQ) mechanism, which retransmits extra bits to correct erroneous bits obtained after FEC decoding. The extra transmitted bits' length equals twice the number of erroneous bits. *LLDPC* provides a lower post-decoding BER, so our system can achieve more robust communication with much shorter re-transmitted bits. LoRa nodes are powered by two AA batteries whose capacity is 3, 000 mAh and send 32-byte payload once per 10 minutes. Note that we do not consider the energy consumption of waiting for the ACK. According to the LoRaWAN specification, LoRa nodes must receive an ACK in the first or second receiving window before transmitting the next packet. They will be opened at a delay of one and two seconds after the end of the uplink. Therefore, whether it is *LLDPC* or Hamming codes, the MCU of LoRa nodes needs to wait the same amount of time with the same power. They will consume the same energy to wait for the ACK.

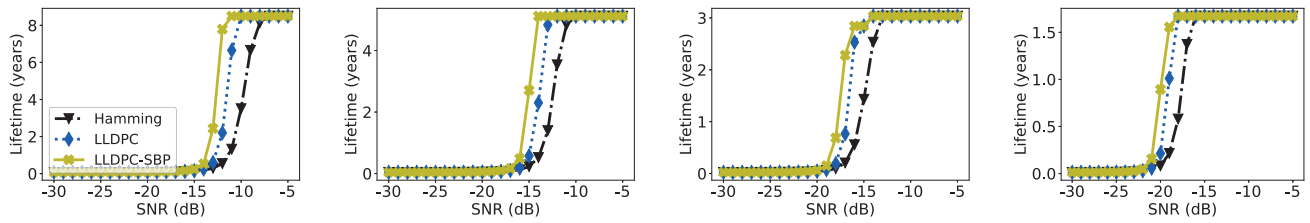
6.1.2 Benchmarks. We compare the performance of *LLDPC* with the following two baselines.

Hamming [1]. Hamming codes are the standard FEC codes for LoRa. LoRa uses k/n Hamming codes with $k = 4$ and $n \in \{5, 6, 7, 8\}$, where k denotes the data word length and n is the codeword length. Hamming codes have a limited ability to detect and correct erroneous bits [3]. The 4/5 and 4/6 CRs can only detect errors, while the 4/7 and 4/8 CRs can correct one erroneous bit per n bits.



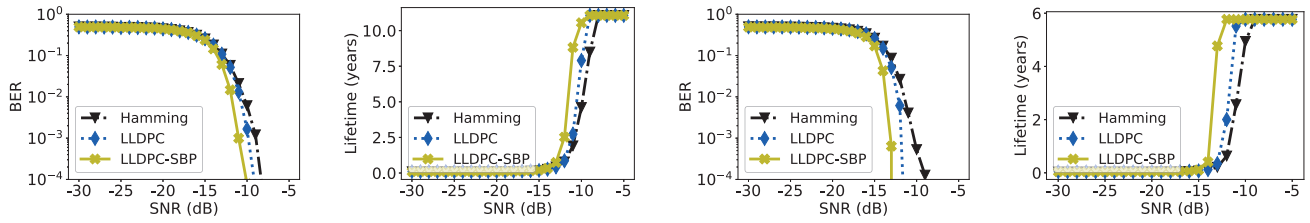
(a) BER *vs.* SNR when SF = 7. (b) BER *vs.* SNR when SF = 8. (c) BER *vs.* SNR when SF = 9. (d) BER *vs.* SNR when SF = 10.

Figure 10: The bit error ratio curves of three decoding methods for different SFs (coding rate: 4/6).



(a) Lifetime *vs.* SNR when SF = 7. (b) Lifetime *vs.* SNR when SF = 8. (c) Lifetime *vs.* SNR when SF = 9. (d) Lifetime *vs.* SNR when SF = 10.

Figure 11: The lifetime curves of three decoding methods for different SFs (coding rate: 4/6).



(a) BER *vs.* SNR when CR = 4/5. (b) Lifetime *vs.* SNR when CR = 4/5. (c) BER *vs.* SNR when CR = 4/7. (d) Lifetime *vs.* SNR when CR = 4/7.

Figure 12: The bit error ratio and lifetime curves of three decoding methods for different coding rates when SF = 7.

LLDPC-SBP. We also implement the SBP algorithm to perform LDPC decoding, as introduced in Section 2.3. We set the maximum iteration of *LLDPC*-SBP is 10, 000.

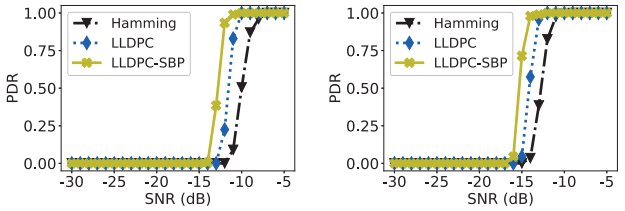
6.2 Overall Performance

We evaluate the performance of *LLDPC* with various LoRa configurations, including 4 SFs (*i.e.*, 7, 8, 9, 10) and 3 CRs (*i.e.*, 4/5, 4/6, and 4/7). We define the SNR threshold as the SNR when the BER is $1e^{-4}$ on the BER-SNR curve. Because this BER can achieve a packet delivery rate of 99.68% when the packet size is 40 bytes. The results are shown in Figures 10, 11, 12, and 13.

BER. Figure 10 shows the performance of different SFs on BER for SNR levels of $[-30, -5]$ dB with a CR of 4/6. We can observe that *LLDPC* obtained consistently lower BER than Hamming codes from SF7 to SF10 across all SNR levels. The SNR threshold of *LLDPC* at an SF can almost catch up with Hamming codes at a higher SF. For example, comparing Figure 10(a) with Figure 10(b), the SNR threshold of *LLDPC* at SF7 is close to Hamming codes' at SF8. Specifically, *LLDPC* lowers the SNR threshold by 2.6, 2.3, 2.3, and 2.2 dB from SF7 to SF10 compared to Hamming codes. *LLDPC*-SBP performs the lowest BERs. This is because *LLDPC*-SBP performs LDPC decoding with the SBP algorithm, which iterates many times to correct erroneous bits. However, the cost is the high decoding latency. *LLDPC* trades off the BER and decoding latency by adopting the GNN-based BP algorithm. Our system can also switch to the *LLDPC*-SBP for lower BER performance if applications relax the decoding latency requirement, *e.g.*, ACK is unnecessary.

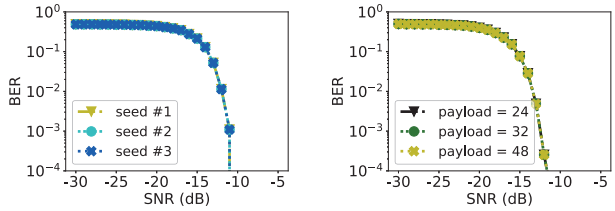
Lifetime. We further evaluate the lifetime of *LLDPC* at different LoRa settings and SNR levels, as shown in Figure 11. We can see that *LLDPC* achieves a higher lifetime than Hamming codes, resulting in a consistent lifetime gain. Compared to the Hamming codes, *LLDPC* extends the median battery life by 51.2%, 31.2%, 13.9%, and 21.2% from SF7 to SF10. Additionally, Figure 11(d) shows that SF10 provides a lifetime of fewer than two years, much shorter than SF7 because SF10 uses longer symbols than SF7.

Coding Rate. We vary the CRs to examine the effect of different CRs on the *LLDPC*'s performance. Figure 12 displays that as CR increases, the SNR threshold decreases, indicating an increase in



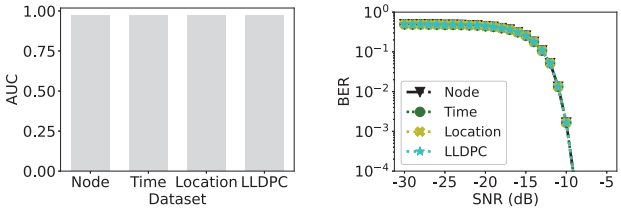
(a) PDR *vs.* SNR when SF = 7. (b) PDR *vs.* SNR when SF = 8.

Figure 13: The Packet Delivery Ratio (PDR) performance evaluation under two SFs when CR = 4/6.



(a) Different parity-check matrices. (b) Different payload length.

Figure 14: The performance of *LLDPC* under different parity-check matrices and payload length.



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

Figure 15: Cross-domain transfer ability analysis.

error correction capability. Figure 12(c) reveals that the SNR threshold for Hamming codes with a CR of 4/7 is also reduced compared to the CRs of 4/5 and 4/6. This is because Hamming codes with a CR of 4/7 can correct one erroneous bit per 7-bit codeword.

Packet Delivery Ratio (PDR). We also investigate PDR [41] for different SFs at SNR levels of $[-30, -5]$ dB with a CR of 4/6. For a particular SNR, the PDR is the ratio of the number of correctly decoded packets divided by the number of all packets transmitted at that SNR. From Figure 13, we can see that regardless of the SFs or SNRs, *LLDPC* still provides the highest PDR, which is consistent with the conclusion obtained from the BER-SNR curves in Figure 10.

6.3 Performance under Different Settings

The above experiment results prove the effectiveness of *LLDPC*. We further investigate whether the performance of *LLDPC* is sensitive to the parity-check matrix and payload length.

6.3.1 Parity-Check Matrix. With different seeds, we can obtain different parity-check matrices for a specific CR, *i.e.*, the entries of 1 at rows and columns are changed. We change the parity-check matrix to investigate whether different matrices impact the performance of *LLDPC*. We randomly generate three parity-check matrices using three seeds, where SF is set to 7 and CR is 4/6. Note that we need to retrain GNN models for different matrices. Figure 14(a) shows that all the seeds achieve the same BER-SNR curves. It verifies that parity-check matrices generated from different seeds do not affect the performance of *LLDPC*.

6.3.2 Payload Length. Given that different payload sizes generate parity-check matrices of different dimensions, we vary the payload size to evaluate the performance of *LLDPC*. We change the payload size to 24, 32, and 48 bytes. The SF is 8, and CR is 4/5. The corresponding dimensions of the parity-check matrix are 48×240, 64×320, and 96×480, respectively. We need to retrain GNN models for different payload sizes because the parity-check matrix is changed. Figure 14(b) demonstrates that payload size does not affect the BER-SNR curves.

6.4 Scalability of *LLDPC*

The SVM and GNN models used in *LLDPC* need to be trained beforehand. We will evaluate whether the performance of the two models is sensitive to the training data.

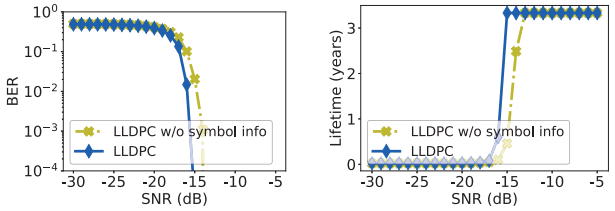
We divide the dataset introduced in Section 4 into multiple sub-datasets based on different scenarios, *i.e.*, LoRa node, location, and time. We use one sub-dataset to generate synthetic training data and other sub-datasets to evaluate the performance of our SVM and GNN models. For example, we collect a synthetic sub-dataset at 11:00 AM from node A in room R1 for training SVM and GNN models. Then we test the trained model on the three datasets. The first sub-dataset is collected at 11:00 AM from Node B in Room R1 (marked as "Node"). The packets sent from node A in room R1 at different times are the second sub-dataset (marked as "Time"). The third sub-dataset is transmitted from Node A in Room R2 at 11:00 AM (marked as "Location").

Figure 15 shows the results. We can see that the SVM model performs the same AUC (Area under the ROC Curve) on the three sub-datasets. Their BER-SNR curves of *LLDPC* with the other three datasets are overlapped with each other. This is because our models are trained with all the different SNRs in a fine-grained way. The SNR is used to quantify the channel quality. Thus, our SVM and GNN models are agnostic to node, time, and location.

6.5 Analysis of *LLDPC* Components

In this section, we evaluate the effectiveness of the proposed two components in *LLDPC*.

6.5.1 Spectrum-based LLR Extractor. To verify the effectiveness of the LLR extractor, we compare the output difference between the LoRa CSS demodulation and our LLR extractor. Specifically, a positive or negative LLR can be used to determine whether a bit is 1 or 0, as introduced in Section 2.3. If the two methods have different values at the same bit position, the difference is incremented by 1. We compare the difference of millions of packets for different configurations (*e.g.*, SFs and CRs) at different SNR levels. We observe that the difference is always zero regardless of SFs, CRs, or SNRs. It



(a) BER. (b) Lifetime.

Figure 16: *LLDPC* with and without symbol-level information when SF = 8, CR = 4/7.

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%

indicates that the LLR extractor does not increase erroneous bits, but provides more information, *i.e.*, the confidence of bit values.

6.5.2 Symbol-Aware LLR Enhancement. We first show confusion matrices to verify the effectiveness of our proposed symbol features and SVM models. To obtain the performance gain of symbol-aware LLR enhancement, we perform LDPC decoding with and without this enhancement using a trained GNN model.

Results. Table 3 exhibits a confusion matrix for SF7 at different SNR levels. We can find that the false positive ratio is low, *i.e.*, 0.3%, and the false negative ratio is higher than the false positive ratio. This is because the features of incorrect symbols differ from correct ones, but some correct ones have similar features to incorrect ones.

We then use the LLR without and with symbol-level information to perform LDPC decoding with trained GNN models. From Figure 16, we can find that the enhanced LLR could decrease the median BER by 27.3% at the SNR range of $[-30, -5]$ dB. Correspondingly, it increases the median lifetime by 47.7%. It further sustains the effectiveness of our symbol-aware LLR enhancement module.

6.6 Campus-Scale Testbed Experiments

In-Field Experimental Design. There is only additive white Gaussian noise in the synthetic dataset used in the previous evaluation section. However, there are other types of noise in actual packets, such as multipath interference. Additionally, LoRa uses an unlicensed spectrum, making it inevitable that multiple networks sharing the same sub-GHz unlicensed band will be co-located. This will lead to cross-technology interference. To examine *LLDPC* on real datasets, we deploy LoRa nodes at different locations covering various land cover types (*e.g.*, ponds, trees, and buildings).

Next, we set SFs and CRs to suit the link quality. During the real data collection, we also record the SNRs of the received packets, reflecting the link quality. Thus, we can mimic LoRa links whose SNR changes with the recorded SNRs. The link adaptation is conducted on this simulated link.

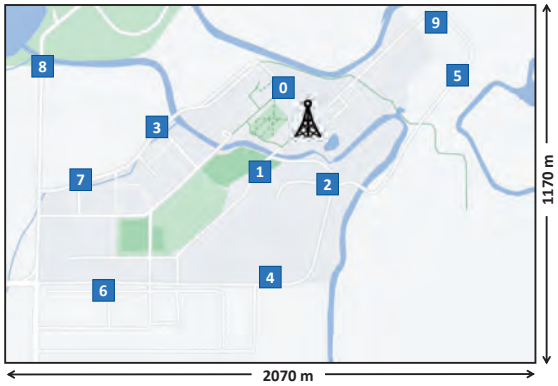
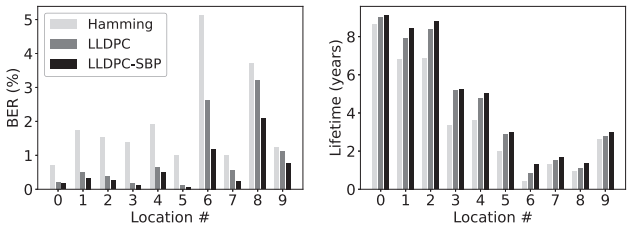


Figure 17: The illustration of our in-field testbed and the topology of the LoRa nodes and the receiver.



(a) BER. (b) Lifetime.

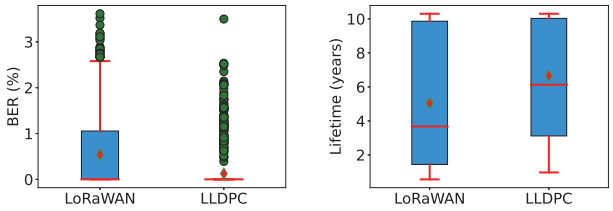
Figure 18: BER and lifetime performance at ten different locations on a campus-scale testbed.

Testbed. In in-filed experiments, we deploy LoRa nodes at ten locations, as shown in Figure 17. Location #0 is the closest, and location #8 is the farthest. The SF settings at ten locations are as follows. At locations #0, #1, and #2, we use SF7. SF8 is selected at locations #3 and #4. We use SF9 to send packets at locations #5 and #9. At locations #6, #7, and #8, SF10 is used. Each LoRa node transmits 150 packets with an interval of 10s. The payload size is 32 bytes, and the CR is 4/5. Hence, the packet size is 40 bytes. Nodes are equipped with a 3,000mAh power bank.

6.6.1 Performance on the Real Dataset. We compute the BER and lifetime for each LoRa node.

Figure 18 shows the BER and the lifetime of Hamming codes, *LLDPC*, and *LLDPC*-SBP. On average, *LLDPC* provides a substantial performance improvement over the Hamming codes on the BER and lifetime. In particular, *LLDPC* can reduce the average BER of Hamming codes among ten nodes by 50.8% and extends the average lifetime by 19.3%.

In Figure 18(a), *LLDPC* has a smaller BER than Hamming codes at all locations, *e.g.*, *LLDPC* decreases the BER at location #6 by 54.5% to 2.63%, and *LLDPC*-SBP further reduces the BER by 48.8% to 1.19%. Compared with location #8, location #6 is near the receiver. But location #6 has a higher BER with Hamming codes due to a low SNR level incurred by the blockage of trees and buildings. We also estimate the lifetime of LoRa nodes at each location. Figure 18(b) displays that the lifetime can be extended by 3.38 to 5.25 years with *LLDPC*. The maximum lifetime can reach 9.08 years by using *LLDPC*-SBP at location #0. Compared with Hamming codes, *LLDPC*



(a) BER. (b) Lifetime.

Figure 19: Joint selection of SF and CR for link adaption.

lowers the BER and extends the lifetime significantly with a CR of 4/5. On the other hand, we observe that the performance of LLPDC in the natural environment degrades by 7.8% compared to the performance in the synthetic dataset. In the future, we could use the real collected packets to fine-tune our SVM and GNN models to adapt to the natural environment.

If we increase CR (*e.g.*, 4/6), the performance gain of *LLDPC* is more significant than the CR of 4/5. This is because the 4/6 Hamming codes cannot correct any erroneous bits [3]; it just adds the overhead of more parity-check bits. But *LLDPC* provides a more powerful error correction ability.

6.6.2 Integration of LLDPC into LoRaWAN. We utilize the simulated links to configure SFs and CRs jointly. Specifically, we use the GitHub library [42] to randomly generate LoRa packets with specific SF and CR determined by the ADR mechanism introduced in Section 5. The library is widely used to generate LoRa packets in the current work [34, 43, 44]. Then packets are transmitted in the additive white Gaussian noise channels with an SNR obtained in the collected SNRs. We use the three recently received packets to predict the SNR of the subsequent transmission. In practice, packets are sent with a low-duty cycle, *e.g.*, 10 minutes. Since nodes send packets with an interval of 10s, we can receive 60 packets per 10 minutes. Hence, 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 19(a) shows the lifetime of LoRaWAN and *LLDPC*. We can find that *LLDPC* can extend the lifetime of nodes by 86.7% on average, compared with LoRaWAN. The reasons are twofold. The link quality varies dynamically from -4dB to -15dB, making the SNR prediction inaccurate. However, *LLDPC* provides a high error correction ability to handle inaccurate SNR prediction. From Figure 19(b), we can see that *LLDPC* reaches a lower BER than LoRaWAN by 97.1%. Second, *LLDPC* can achieve robust communication with smaller SF than LoRaWAN. For the same link quality, *LLDPC* prefers to use smaller SFs than LoRaWAN. Compared to the standard LoRa PHY, *LLDPC* can achieve lower SNR thresholds with the same SF and CR. Therefore, for a given SNR, *LLDPC* can achieve the same BER with smaller SF and CR, which can significantly extend the node lifetime.

6.7 Storage Overhead and Running Time

In *LLDPC*, we need to consider the overhead of the two parts, *i.e.*, gateways and LoRa nodes.

Table 4: Time consumption (unit: s) of GNN-based BP, SBP, and *LLDPC* to decode one packet for different CRs on two platforms (PC/Raspberry Pi 3) when SF = 9. Note that GPU is disabled while performing LDPC decoding online on the PC.

	CR = 4/5	CR = 4/6	CR = 4/7
SBP	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

Table 5: Storage and energy consumption of *LLDPC* on LoRa nodes for MCU and Radio, where the 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

6.7.1 Gateways. We run our SVM and GNN models on a local PC and a single-board computer of Raspberry Pi 3. We measure the storage overhead and running time. The running time measures the time used to decode one packet and is computed by running thousands of times and taking the average. Since the gateway is grid-powered, we do not consider its energy consumption.

Results. The storage overheads of SVM plus GNN models are 15.7, 16.3, and 16.8 MB for CRs of 4/5, 4/6, and 4/7. We find that the model size increases as CRs get larger. This is because a larger CR means a larger packet size, which requires a larger model and more parameters to handle inputs.

According to the LoRaWAN specification, LoRa nodes must receive an ACK in the first or second receiving window before transmitting the next packet. These two windows will be opened at a delay of one and two seconds after the end of the uplink. Given this constraint, Table 4 further displays the running time of the SBP algorithm, GNN-based BP, and the whole system. Similar to storage overhead, running time increases with CRs. It shows that our GNN-based BP algorithm reduces the average decoding latency of Soft BP by 58.09×. We also measure the sum of the elapsed time of *LLDPC*, including the spectrum-based LLR extractor, the symbol-aware LLR enhancement module, and the GNN model. With a powerful computing computer, it takes less than 100 ms. On the Raspberry Pi 3, the consumed time is about 1.056s with a deviation of 0.582s. For CRs of 4/5 and 4/6, the entire running time is less than 1s, which allows the gateway to send ACKs within the first receiving window of the LoRa node. *LLDPC* consumes more than one second when CR is 4/7. Thus, we send ACKs at the second receiving window with a time limit of two seconds.

6.7.2 LoRa Nodes. In *LLDPC*, the overhead of nodes comes from two parts, *i.e.*, storing the parity-check matrix and generating encoded bits. For different CRs, we need to use different parity-check matrices. By extending an SD card module on the Arduino board, we can store parity-check matrices with CR of 4/5, 4/6, and 4/7 in the Arduino board. To generate encoded bits, we first read the

parity-check matrix stored on the SD card module row by row. Then we perform XOR operations to encode the data.

Results. Table 5 shows the storage overheads of parity-check matrices under different CRs. We can see that the size of the matrix increases as the CRs go up. This is because the larger CR has a larger number of rows.

The energy consumption of LDPC encoding and transmitting is also shown in Table 5. As CRs increase, energy consumption also increases. During the encoding process, the MCU is active with a power of 23.48mW [38]. Therefore, such the encoding process consumes energy of 0.9 mJ (23.48mW $\times$ 42.5ms) when CR is set to 4/5. It shows that for a node with a battery capacity of 3000 mAh and a voltage of 1.5 V, the encoding process consumes only 0.6E-5% of the energy. It is almost negligible compared to the energy consumed in the transmission process.

7 RELATED WORK

Error Correction in LoRa Networks. Error correction is widely adopted in wireless networks [45–49]. In WiFi and cellular networks, multiple antennas are used to improve the SNRs of the received signal. Recent studies for LoRa [3, 39, 40, 50, 51] borrow this idea. Choir [50] enhances the SNRs of the received signal by deploying multiple co-located LoRa nodes. Charm [39] decodes weak signals by organizing multiple gateways to detect combined energy peaks in the spectrum. OPR[3] utilizes disjoint link layer bit errors received by multiple gateways to detect erroneous bits and correct them using the CRC defined in the MAC layer. Nephalai [51] transmits compressed PHY samples to the cloud and demodulates compressed PHY samples with the sparse approximation. In addition, Chime [40] tries to avoid multipath interference via choosing the operating frequency of LoRa nodes, obtaining extra SNR gain for LoRa transmission. All of these systems require multiple pairs of transceivers. In contrast, *LLDPC* achieves additional SNR gain through the proposed GNN-based BP algorithm to perform LDPC decoding using only one pair of transceivers. Therefore, *LLDPC* can supplement existing works and be further enhanced by the diversity gain of multiple gateways.

Elshabrawy *et al.* [52] theoretically analyze the benefits of adopting Non-Binary Single Parity-Check Codes. However, the encoding process of the Non-Binary Single Parity-Check Codes requires the arithmetic calculation in the GF, which is computation-expensive [8]. In contrast, *LLDPC* uses binary LDPC codes, which involve only simple XOR operations. These XOR operations can be done at the sensor node without consuming much energy. In addition, we propose a novel encoding scheme, the design, and the implementation of the architecture based on LDPC codes.

LoRa Throughput. FTrack [53] NSacle [35], CoLoRa [54], CurvingLoRa [55], and PCube [56] develop several methods to resolve the packet collisions in LoRa, thereby increasing the throughput of the LoRa network [57]. In contrast, *LLDPC* developed a system that integrates LDPC codes into LoRa to extend the communication range of LoRa nodes and optimize energy efficiency, paralleling research in this category.

NELoRa [34] proposed a neural-enhanced LoRa demodulation method, which exploits the feature abstraction ability of deep learning to support LoRa demodulation under ultra-low SNR. *LLDPC* targets a different task, *i.e.*, FEC coding. Both are necessary components of LoRa physical layer. *LLDPC* extracts LLRs by the amplitude spectrum from CSS modulation with the SoftMax operation (Equation 5). Our LLR extractor can also use the output of NELoRa to obtain the LLR of bits. This is because NELoRa can output the probabilities of all frequency bins based on its final SoftMax layer. Therefore, NELoRa and *LLDPC* could work together for a more robust LoRa communication.

Graph Neural Network-based BP Algorithm. Deep learning-based methods are proposed for various applications, such as wireless networking [34], smartwatch-based arm tracking [58], and smartphone apps usage prediction [59, 60]. The belief propagation algorithm performs iterative message passing between bit nodes and factor nodes on factor graphs. Researchers are trying to implement message exchange using GNN models [16, 61]. NEBP [61] designed a GNN model to operate on the factor graph and exchange information with traditional BP algorithm for error correction decoding, which is still time-consuming. Therefore, it cannot be used in the LoRaWAN. FGNN [16] generalizes graph neural networks to factor graph neural networks, which allows the network to capture higher-order dependencies among bit nodes and factor nodes. *LLDPC* leverages large-scale synthetic LoRa packets to generate the required feature for the GNN model, then selects the appropriate number of GNN layers empirically to perform fast LDPC decoding.

8 CONCLUSION

This paper presents *LLDPC*, an efficient error correction coding scheme for LoRa networks. We integrate LDPC codes in LoRa by combining a spectrum-based LLR extractor, a symbol-aware LLR enhancement module, and a GNN-based belief propagation algorithm. *LLDPC* can provide exceptional error correction capability compared with standard Hamming codes in LoRa. Extensive simulation on the large-scale synthetic dataset and in-field experiment demonstrate the effectiveness of *LLDPC*.

ACKNOWLEDGEMENTS

We would like to thank our anonymous shepherd and reviewers for their constructive comments. This research was partially supported by a Fresno-Merced Future of Food Innovation Initiative (F3) challenge grant, the UC National Laboratory Fees Research Program grant #69763, and a 2022 Faculty Research Award through the Academic Senate Faculty Research Program at UC Merced.

REFERENCES

[1] LoRaWAN^{TM} 1.1 Specification . https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf, 2017.
[2] Jothi Prasanna Shanmuga Sundaram, Wan Du, and Zhiwei Zhao. A survey on LoRa networking: Research problems, current solutions, and open issues. *IEEE Communications Surveys & Tutorials*, 22(1):371–388, 2019.
[3] Artur Balanuta, Nuno Pereira, Swarun Kumar, and Anthony Rowe. A cloud-optimized link layer for low-power wide-area networks. In *ACM MobiSys*, 2020.
[4] SX1272/3/6/7/8: LoRa Modem Design Guide. https://www.openhacks.com/uploadsproductos/loradesignguide_std.pdf, 2013.
[5] Ian F Akyildiz, Tommaso Melodia, and Kaushik R Chowdury. Wireless multimedia sensor networks: A survey. *IEEE Wireless Communications*, 14(6):32–39, 2007.
[6] Wan Du, Zhenjiang Li, Jansen Christian Liando, and Mo Li. From rateless to distanceless: Enabling sparse sensor network deployment in large areas. In *ACM SenSys*, 2014.
[7] Wei Song, Yifei Shen, Liping Li, Kai Niu, and Chuan Zhang. A general construction and encoder implementation of polar codes. *IEEE Transactions on Very Large Scale Integration Systems*, 28(7):1690–1702, 2020.

[8] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[9] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[10] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[11] 5G; NR; Multiplexing and channel coding. https://www.etsi.org/deliver/etsi_ts/138200_138299/138212/15.02.00_60/ts_138212v150200p.pdf, 2018.

[12] Alberto Morello and Vittoria Mignone. DVB-S2: The second generation standard for satellite broad-band services. *Proceedings of the IEEE*, 94(1):210–227, 2006.

[13] 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 802.11-2012 (Revision of IEEE Std 802.11-2007)*, 2012.

[14] Fatma Benkhelifa, Yathreb Bouazizi, and Julie A McCann. How Orthogonal is LoRa Modulation? *IEEE Internet of Things Journal*, 2022.

[15] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[16] Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. In *NeurIPS*, 2020.

[17] LLDPC's code. https://github.com/kangyang73/LLDPC-SenSys22, 2022.

[18] R Tanner. A recursive approach to low complexity codes. *IEEE Transactions on information theory*, 27(5):533–547, 1981.

[19] Arduino Uno Rev3. https://store-usa.arduino.cc/products/arduino-uno-rev3/?selectedStore=us, 2021.

[20] Juntan Zhang and Marc PC Fossorier. A modified weighted bit-flipping decoding of low-density parity-check codes. *IEEE Communications Letters*, 8(3):165–167, 2004.

[21] Jianguang Zhao, Farhad Zarkeshvari, and Amir H Banihashemi. On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes. *IEEE transactions on communications*, 53(4):549–554, 2005.

[22] Rolf Johannesson and Kamil Sh Zigangirov. *Fundamentals of convolutional coding*. John Wiley & Sons, 2015.

[23] Eugenio Magistretti, Krishna Kant Chintalapudi, Bozidar Radunovic, and Ramachandran Ramjee. WiFi-Nano: Reclaiming WiFi efficiency through 800 ns slots. In *ACM MobiCom*, 2011.

[24] Yong Soo Cho, Jaekwon Kim, Won Y Yang, and Chung G Kang. *MIMO-OFDM wireless communications with MATLAB*. John Wiley & Sons, 2010.

[25] Wan Du, Jansen Christian Liando, and Mo Li. Softlight: Adaptive visible light communication over screen-camera links. In *IEEE INFOCOM*, 2016.

[26] Wan Du, Jansen Christian Liando, and Mo Li. Soft hint enabled adaptive visible light communication over screen-camera links. *IEEE Transactions on Mobile Computing*, 16(2):527–537, 2017.

[27] Srikar Kasi and Kyle Jamieson. Towards quantum belief propagation for LDPC decoding in wireless networks. In *ACM MobiCom*, 2020.

[28] Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. ATPP: A mobile app prediction system based on deep marked temporal point processes. In *IEEE DCOSS*, 2021.

[29] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[30] Huatao Xu, Pengfei Zhou, Rui Tan, Mo Li, and Guobin Shen. LIMU-BERT: Unleashing the Potential of Unlabeled Data for IMU Sensing Applications. In *ACM SenSys*, 2021.

[31] Semtech SX1276 datasheet. https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276, 2020.

[32] GR-LoRa. https://github.com/rpp0/gr-lora, 2021.

[33] Muhammad Osama Shahid, Millan Philipose, Krishna Chintalapudi, Suman Banerjee, and Bhuvana Krishnaswamy. Concurrent interference cancellation: decoding multi-packet collisions in LoRa. In *ACM SIGCOMM*, 2021.

[34] Chenning Li, Hanqing Guo, Shuai Tong, Xiao Zeng, Zhichao Cao, Mi Zhang, Qiben Yan, Li Xiao, Jiliang Wang, and Yunhao Liu. NELoRa: Towards Ultra-low SNR LoRa Communication with Neural-enhanced Demodulation. In *ACM SenSys*, 2021.

[35] Shuai Tong, Jiliang Wang, and Yunhao Liu. Combating packet collisions using non-stationary signal scaling in LPWANs. In *ACM MobiSys*, 2020.

[36] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Jansen C Liando, Amalinda Gamage, Agustinus W Tengourtius, and Mo Li. Known and unknown facts of LoRa: Experiences from a large-scale measurement study. *ACM Transactions on Sensor Networks*, 15(2):1–35, 2019.

[39] Adwait Dongare, Revathy Narayanan, Akshay Gadre, Anh Luong, Artur Balanuta, Swarun Kumar, Bob Iannucci, and Anthony Rowe. Charm: exploiting geographical diversity through coherent combining in low-power wide-area networks. In *ACM/IEEE IPSN*, 2018.

[40] Akshay Gadre, Revathy Narayanan, Anh Luong, Anthony Rowe, Bob Iannucci, and Swarun Kumar. Frequency Configuration for Low-Power Wide-Area Networks in a Heartbeat. In *USENIX NSDI*, 2020.

[41] Weifeng Gao, Wan Du, Zhiwei Zhao, Geyong Min, and Mukesh Singhal. Towards energy-fairness in LoRa networks. In *IEEE ICDCS*, 2019.

[42] LoRaPHY. https://github.com/jkadbear/LoRaPHY, 2022.

[43] Jinyan Jiang, Zhenqiang Xu, Fan Dang, and Jiliang Wang. Long-range ambient LoRa backscatter with parallel decoding. In *ACM MobiCom*, 2021.

[44] Shuai Tong, Zilin Shen, Yunhao Liu, and Jiliang Wang. Combating link dynamics for reliable LoRa connection in urban settings. In *ACM MobiCom*, 2021.

[45] Wan Du, Zhenjiang Li, Jansen Christian Liando, and Mo Li. From rateless to distanceless: Enabling sparse sensor network deployment in large areas. *IEEE/ACM Transactions on Networking*, 24(4):2498–2511, 2015.

[46] Aditya Gudipati and Sachin Katti. Strider: Automatic rate adaptation and collision handling. In *ACM SIGCOMM*, 2011.

[47] Wan Du, Jansen Christian Liando, Huanle Zhang, and Mo Li. When pipelines meet fountain: Fast data dissemination in wireless sensor networks. In *ACM SenSys*, 2015.

[48] Binbin Chen, Ziling Zhou, Yuda Zhao, and Haifeng Yu. Efficient error estimating coding: Feasibility and applications. In *ACM SIGCOMM*, 2010.

[49] Wan Du, Jansen Christian Liando, Huanle Zhang, and Mo Li. Pando: Fountain-enabled fast data dissemination with constructive interference. *IEEE/ACM Transactions on Networking*, 25(2):820–833, 2017.

[50] Rashad Eletreby, Diana Zhang, Swarun Kumar, and Osman Yağan. Empowering low-power wide area networks in urban settings. In *ACM SIGCOMM*, 2017.

[51] Jun Liu, Weitao Xu, Sanjay Jha, and Wen Hu. Nephalai: towards LPWAN C-RAN with physical layer compression. In *ACM MobiCom*, 2020.

[52] Tallal Elshabrawy and Joerg Robert. Enhancing LoRa capacity using non-binary single parity check codes. In *IEEE WiMob*, 2018.

[53] Xianjin Xia, Yuanqing Zheng, and Tao Gu. FTrack: Parallel decoding for LoRa transmissions. In *ACM SenSys*, 2019.

[54] Shuai Tong, Zhenqiang Xu, and Jiliang Wang. CoLoRa: Enabling multi-packet reception in LoRa. In *IEEE INFOCOM*, 2020.

[55] Chenning Li, Xiuzhen Guo, Longfei Shangguan, Zhichao Cao, and Kyle Jamieson. CurvingLoRa to Boost LoRa Network Throughput via Concurrent Transmission. In *USENIX NSDI*, 2022.

[56] Xianjin Xia, Ningning Hou, Yuanqing Zheng, and Tao Gu. PCube: scaling LoRa concurrent transmissions with reception diversities. In *ACM MobiCom*, 2021.

[57] Amalinda Gamage, Jansen Christian Liando, Chaojie Gu, Rui Tan, and Mo Li. LMAC: Efficient carrier-sense multiple access for LoRa. In *ACM MobiCom*, 2020.

[58] Miaomiao Liu, Sikai Yang, Wyssanie Chomsin, and Wan Du. Real-Time Tracking of Smartwatch Orientation and Location by Multitask Learning. In *ACM SenSys*, 2022.

[59] Zhihao Shen, Kang Yang, Zhao Xi, Jianhua Zou, and Wan Du. DeepAPP: A Deep Reinforcement Learning Framework for Mobile Application Usage Prediction. *IEEE Transactions on Mobile Computing*, pages 1–1, 2021.

[60] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, and Jianhua Zou. DeepAPP: A Deep Reinforcement Learning Framework for Mobile Application Usage Prediction. In *ACM SenSys*, 2019.

[61] Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *AISTATS*, 2021.