## A. The parity check matrix used in our neural decoder and the permutations used in the list decoding algorithm

We first describe how to produce the generator matrix and the parity check matrix for BCH codes and punctured RM codes with code length $n = 2^m - 1$ in our decoding algorithm.
**Step 1**: Find a primitive element $\alpha$ of the finite field $\mathbb{F}_{2^m}$. Then the elements of this finite field are $0, 1, \alpha, \alpha^2, \dots, \alpha^{2^m-2}$.
**Step 2**: Find the generator polynomial of the code. For $1 \le j \le 2^m - 2$, let $M^{(j)}(x)$ be the minimal polynomial of $\alpha^j$ over the binary field. For BCH code with designed distance $2\delta + 1$, the generator polynomial is $g(x) = \mathrm{lcm}\{M^{(1)}(x), M^{(3)}(x), \dots, M^{(2\delta-1)}(x)\}$, where lcm stands for least common multiple; see Chapter 7.6 of (MacWilliams & Sloane, 1977). For $r$th order punctured RM code, the generator polynomial is

$$g(x) = \mathrm{lcm}\{M^{(j)}(x) : 1 \le j \le 2^m - 2,\ 1 \le w_2(j) \le m - r - 1\}, \tag{13}$$

where $w_2(j)$ is the number of 1's in the binary expansion of $j$. For example, $w_2(3) = 2$ because the binary expansion of 3 is $(1, 1)$, and $w_2(5) = 2$ because the binary expansion of 5 is $(1, 0, 1)$. See Theorem 11 in Chapter 13.5 of (MacWilliams & Sloane, 1977) for a proof of (13). For an $(n, k)$ cyclic code, the degree of the generator polynomial is $n - k$, so $g(x)$ can be written as $g(x) = g_0 + g_1x + g_2x^2 + \cdots + g_{n-k}x^{n-k}$, where the coefficients $g_0, g_1, \dots, g_{n-k}$ are either 0 or 1. Then the following $k \times n$ matrix is a generator matrix of the code:

$$\begin{matrix} g_0 & g_1 & g_2 & \dots & g_{n-k} & 0 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & g_2 & \dots & g_{n-k} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & g_0 & g_1 & g_2 & \dots & g_{n-k} \end{matrix} . \tag{14}$$

**Step 3**: For cyclic codes, it always holds that $g(x)$ divides $x^n - 1$, and the parity check polynomial of the code is $h(x) = (x^n - 1)/g(x)$. Since the degree of $g$ is $n - k$, the degree of $h$ is $k$, and so $h(x)$ can be written as $h(x) = h_kx^k + \cdots + h_2x^2 + h_1x + h_0$, where the coefficients $h_k, \dots, h_2, h_1, h_0$ are either 0 or 1. This explains how to obtain $h_k, \dots, h_2, h_1, h_0$ in the parity matrix (9). As already mentioned in Section 3, matrix (9) is used in (Nachmani et al., 2016; 2018). In contrast, we use an $n \times n$ parity check matrix consisting of all the $n$ cyclic shifts of the first row of matrix (9) in our decoder.

Next we describe how to find the permutations used in the list decoding algorithm. More precisely, we explain how to find the set $\mathcal{S}$ consisting of $n + 1 = 2^m$ permutations; see the beginning part of Section 4. Note that for all BCH codes and punctured RM codes with code length $n$, we use the **same** set $\mathcal{S}$ of permutations. In other words, the set $\mathcal{S}$ only depends on the code length $n$, and it does not change with code dimension or other parameters.

It is well known that both RM codes and extended BCH codes are invariant to the affine group (Kasami et al., 1967). Since $\mathcal{S}$ is a subset of this group, let us begin with describing the affine group. Let $\mathcal{C}$ be a BCH code or a punctured RM code with code length $n$, and let $G$ be its generator matrix obtained from the procedure described above, so $G$ is of the form (14). Let $(C_1, C_2, \dots, C_n)$ be a codeword generated from the matrix $G$. Notice that the matrix $G$ specifies the order of coordinates in the codeword. (For example, swapping two columns of $G$ amounts to swapping the two corresponding coordinates in the codeword.) As already mentioned in Section 4, by prepending an overall parity bit $C_0$ we obtain $(C_0, C_1, \dots, C_n)$, a codeword from an extended BCH code or a RM code with length $n + 1$. Next we define a one-to-one mapping $f$ between the index set $\{0, 1, \dots, n\}$ and the finite field $\mathbb{F}_{2^m} = \{0, 1, \alpha, \alpha^2, \dots, \alpha^{n-1}\}$:

$$f(0) = 0, \quad f(i) = \alpha^{i-1} \text{ for } i \in [n].$$

For $a, b \in \mathbb{F}_{2^m}, a \ne 0$, the affine mapping $X \mapsto aX + b$ defines a permutation on the finite field $\mathbb{F}_{2^m}$, and through the function $f$ it also induces a permutation on the index set $\{0, 1, \dots, n\}$. More precisely, for $1 \le i \le n$ and $0 \le j \le n$, we use $\sigma_{i,j}$ to denote the permutation on $\{0, 1, \dots, n\}$ induced by the mapping $X \mapsto f(i)X + f(j)$:

$$\sigma_{i,j}(v) = f^{-1}\big(f(i)f(v) + f(j)\big) \text{ for } v \in \{0, 1, \dots, n\}.$$

The permutations $\{\sigma_{i,j} : 1 \le i \le n, 0 \le j \le n\}$ form the affine group to which the extended code is invariant.

For the special case of $j = 0$, we have

$$\sigma_{i,0}(0) = 0, \quad \sigma_{i,0}(v) = f^{-1}(\alpha^{i+v-2}) = \begin{cases} v + i - 1 & \text{for } 1 \le v \le n - i + 1 \\ v + i - 1 - n & \text{for } n - i + 2 \le v \le n \end{cases} .$$

Therefore, $\sigma_{i,0}$ is the permutation that fixes $C_0$ and performs $(i-1)$ cyclic right shifts on $(C_1, C_2, \dots, C_n)$. It is not a surprise that the extended code is invariant to such a permutation because $(C_1, C_2, \dots, C_n)$ belongs to a cyclic code.

For the purpose of list decoding, we focus on another special case $i = 1$, and we write $\sigma_j = \sigma_{1,j}$ to simplify the notation. By definition, $\sigma_j$ is the permutation on $\{0, 1, \dots, n\}$ induced by the mapping $X \mapsto X + f(j)$, so $\sigma_j(v) = f^{-1}(f(v) + f(j))$ for $0 \le v \le n$. Clearly, $\sigma_0$ is the identity permutation. The set $\mathcal{S}$ used in our list decoding algorithm is $\mathcal{S} = \{\sigma_0, \sigma_1, \dots, \sigma_n\}$. Here we give a concrete example for $n = 15$. Each row in the following matrix represents a permutation $\sigma_j$. From top to bottom, these permutations are $\sigma_0, \sigma_1, \sigma_2, \sigma_5, \sigma_3, \sigma_9, \sigma_6, \sigma_{11}, \sigma_4, \sigma_{15}, \sigma_{10}, \sigma_8, \sigma_7, \sigma_{14}, \sigma_{12}, \sigma_{13}$.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 0 | 5 | 9 | 15 | 2 | 11 | 14 | 10 | 3 | 8 | 6 | 13 | 12 | 7 | 4 |
| 2 | 5 | 0 | 6 | 10 | 1 | 3 | 12 | 15 | 11 | 4 | 9 | 7 | 14 | 13 | 8 |
| 5 | 2 | 1 | 11 | 8 | 0 | 9 | 13 | 4 | 6 | 15 | 3 | 14 | 7 | 12 | 10 |
| 3 | 9 | 6 | 0 | 7 | 11 | 2 | 4 | 13 | 1 | 12 | 5 | 10 | 8 | 15 | 14 |
| 9 | 3 | 11 | 1 | 14 | 6 | 5 | 15 | 12 | 0 | 13 | 2 | 8 | 10 | 4 | 7 |
| 6 | 11 | 3 | 2 | 12 | 9 | 0 | 10 | 14 | 5 | 7 | 1 | 4 | 15 | 8 | 13 |
| 11 | 6 | 9 | 5 | 13 | 3 | 1 | 8 | 7 | 2 | 14 | 0 | 15 | 4 | 10 | 12 |
| 4 | 15 | 10 | 7 | 0 | 8 | 12 | 3 | 5 | 14 | 2 | 13 | 6 | 11 | 9 | 1 |
| 15 | 4 | 8 | 14 | 1 | 10 | 13 | 9 | 2 | 7 | 5 | 12 | 11 | 6 | 3 | 0 |
| 10 | 8 | 4 | 12 | 2 | 15 | 7 | 6 | 1 | 13 | 0 | 14 | 3 | 9 | 11 | 5 |
| 8 | 10 | 15 | 13 | 5 | 4 | 14 | 11 | 0 | 12 | 1 | 7 | 9 | 3 | 6 | 2 |
| 7 | 14 | 12 | 4 | 3 | 13 | 10 | 0 | 11 | 15 | 6 | 8 | 2 | 5 | 1 | 9 |
| 14 | 7 | 13 | 15 | 9 | 12 | 8 | 1 | 6 | 4 | 11 | 10 | 5 | 2 | 0 | 3 |
| 12 | 13 | 7 | 10 | 6 | 14 | 4 | 2 | 9 | 8 | 3 | 15 | 0 | 1 | 5 | 11 |
| 13 | 12 | 14 | 8 | 11 | 7 | 15 | 5 | 3 | 10 | 9 | 4 | 1 | 0 | 2 | 6 |

As a final remark, all the methods described above can be efficiently programmed using the Communications Toolbox of Matlab. The Matlab code is available at github.com/cyclicallyneuraldecoder

## B. Our neural decoder is equivariant to all cyclic shifts

Recall the definition of cyclic shifts $\pi_j, j \in [n]$ in (7). Observe that $(C_{\pi_j(1)}, C_{\pi_j(2)}, \dots, C_{\pi_j(n)})$ is obtained by $(j-1)$ cyclic left shifts of $(C_1, C_2, \dots, C_n)$. In particular, $(C_{\pi_2(1)}, C_{\pi_2(2)}, \dots, C_{\pi_2(n)}) = (C_2, C_3, \dots, C_n, C_1)$ is obtained by one cyclic left shift of $(C_1, C_2, \dots, C_n)$.

Let $(L_1, L_2, \dots, L_n)$ be an LLR vector and let $(o_1, o_2, \dots, o_n)$ be the corresponding decoding result of our neural decoder. We will prove that for every $j \in [n]$, if the LLR vector is $(L_{\pi_j(1)}, L_{\pi_j(2)}, \dots, L_{\pi_j(n)})$, then the decoding result of our decoder becomes $(o_{\pi_j(1)}, o_{\pi_j(2)}, \dots, o_{\pi_j(n)})$. In fact, we only need to prove this claim for $j = 2$, and the claim for other values of $j$ follows by a simple induction. Below we will write $\pi = \pi_2$ to simplify the notation.

Recall that we use (3) to calculate the messages in even layers and we use (11) for odd layers. Finally, we use (12) to calculate the output layer. Also recall that $E$ is the set of edges in the Tanner graph. Let $x^{[s]}(e), e \in E$ be the messages in the $s$th layer when the input LLR vector is $(L_1, L_2, \dots, L_n)$, and let $\tilde{x}^{[s]}(e), e \in E$ be the messages in the $s$th layer when the input LLR vector is $(\tilde{L}_1, \tilde{L}_2, \dots, \tilde{L}_n) = (L_{\pi(1)}, L_{\pi(2)}, \dots, L_{\pi(n)})$. We will prove that

$$\tilde{x}^{[s]}((c_i, v_j)) = x^{[s]}((c_{\pi(i)}, v_{\pi(j)})) \text{ for all } (c_i, v_j) \in E. \tag{15}$$

Notice that whenever $(c_i, v_j) \in E$, we always have $(c_{\pi(i)}, v_{\pi(j)}) \in E$; see Fig. 2 for an illustration.

We prove (15) by induction on $s$. It holds for $s = 0$ because we set the initialization (both $x^{[0]}$ and $\tilde{x}^{[0]}$) to be the all-zero vector in our algorithm. This establishes the induction base.

Now let $s$ be an odd number and suppose that (15) holds for $s-1$. Let us prove that (15) also holds for $s$. By (11),

$$\begin{aligned}\tilde{x}^{[s]}((c_{\pi_j(i_b)}, v_j)) &= \tanh\Big(\frac{1}{2}\Big(w_b^{[s]}\tilde{L}_j + \sum_{b'\in[u]\setminus\{b\}} w_{b',b}^{[s]}\,\tilde{x}^{[s-1]}((c_{\pi_j(i_{b'})}, v_j))\Big)\Big)\\ &= \tanh\Big(\frac{1}{2}\Big(w_b^{[s]}L_{\pi(j)} + \sum_{b'\in[u]\setminus\{b\}} w_{b',b}^{[s]}\,x^{[s-1]}((c_{\pi(\pi_j(i_{b'}))}, v_{\pi(j)}))\Big)\Big)\\ &= x^{[s]}((c_{\pi(\pi_j(i_b))}, v_{\pi(j)})),\end{aligned}$$

where the second equality follows from the induction hypothesis.

Now let $s$ be an even number and suppose that (15) holds for $s-1$. Let us prove that (15) also holds for $s$. By (3),

$$\begin{aligned}\tilde{x}^{[s]}((c_i, v_j)) &= 2\tanh^{-1}\Big(\prod_{(c_i,v_{j'})\in N(c_i)\setminus\{(c_i,v_j)\}} \tilde{x}^{[s-1]}((c_i, v_{j'}))\Big)\\ &= 2\tanh^{-1}\Big(\prod_{(c_i,v_{j'})\in N(c_i)\setminus\{(c_i,v_j)\}} x^{[s-1]}((c_{\pi(i)}, v_{\pi(j')}))\Big)\\ &= 2\tanh^{-1}\Big(\prod_{(c_{\pi(i)},v_{\pi(j')})\in N(c_{\pi(i)})\setminus\{(c_{\pi(i)},v_{\pi(j)})\}} x^{[s-1]}((c_{\pi(i)}, v_{\pi(j')}))\Big)\\ &= x^{[s]}((c_{\pi(i)}, v_{\pi(j)})).\end{aligned}$$

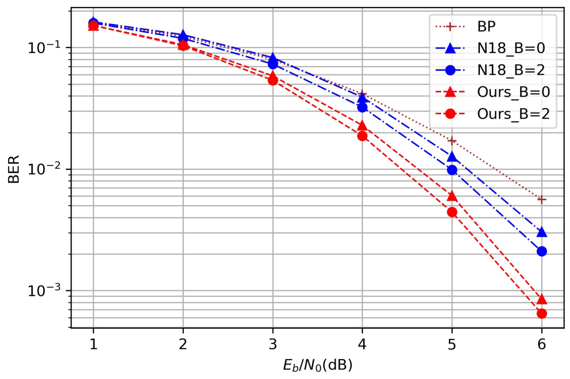
This establishes the inductive step and completes the proof of (15).

Finally, denote the output corresponding to $(\tilde{L}_1, \tilde{L}_2, \dots, \tilde{L}_n)$ as $(\tilde{o}_1, \tilde{o}_2, \dots, \tilde{o}_n)$. Then by (12),

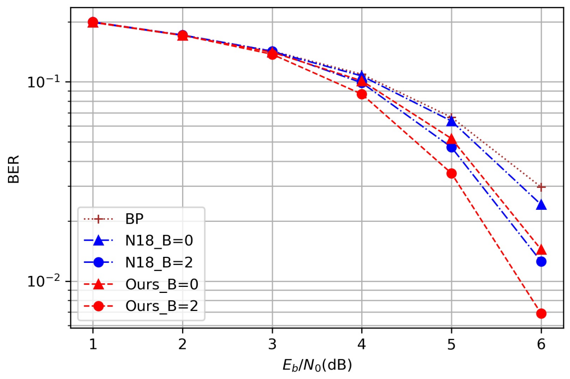
$$\begin{aligned}\tilde{o}_j &= \tilde{L}_j + \sum_{b\in[u]} w_b^{\text{out}}\,\tilde{x}^{[2t]}((c_{\pi_j(i_b)}, v_j))\\ &= L_{\pi(j)} + \sum_{b\in[u]} w_b^{\text{out}}\,x^{[2t]}((c_{\pi(\pi_j(i_b))}, v_{\pi(j)}))\\ &= o_{\pi(j)}.\end{aligned}$$

Thus we have proved that our neural decoder is equivariant to all cyclic shifts.

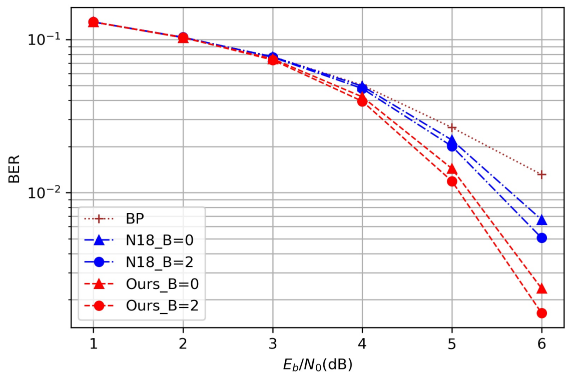
## C. More plots of the simulation results



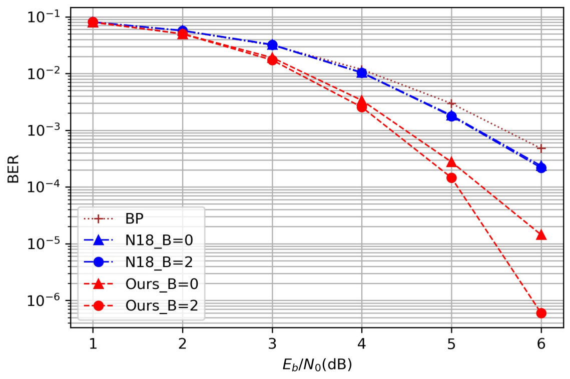
(a) BCH(63,24)



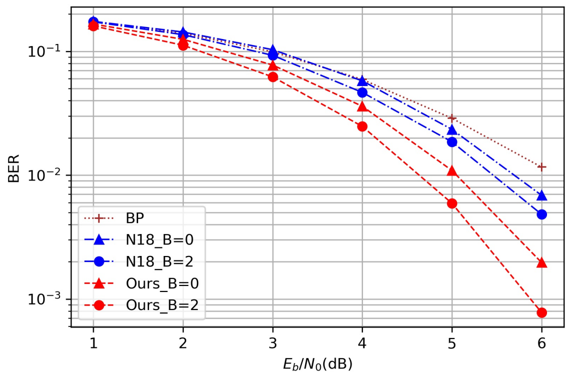
(b) BCH(127,36)



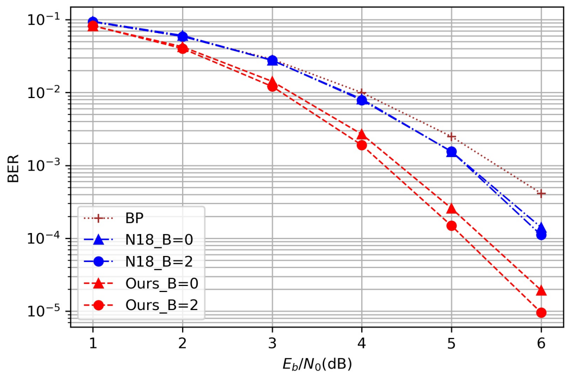
(c) BCH(127,64)



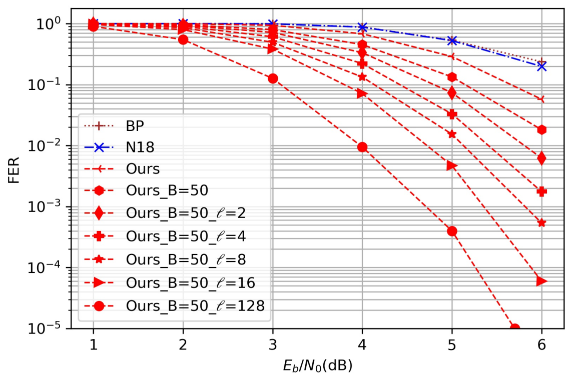
(d) Punctured RM(127,99)



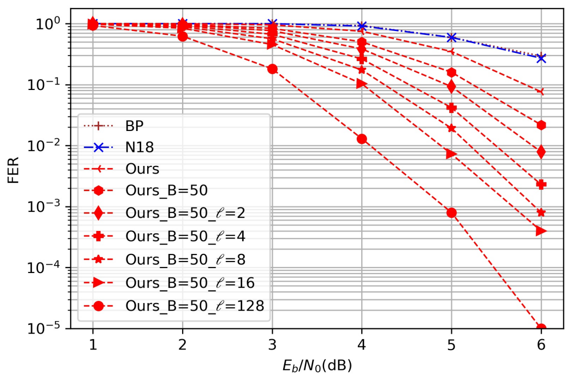
(e) Punctured RM(63,22)



(f) Punctured RM(63,42)



(a) BCH(127,64) List decoding



(b) Punctured RM(127,64) List decoding