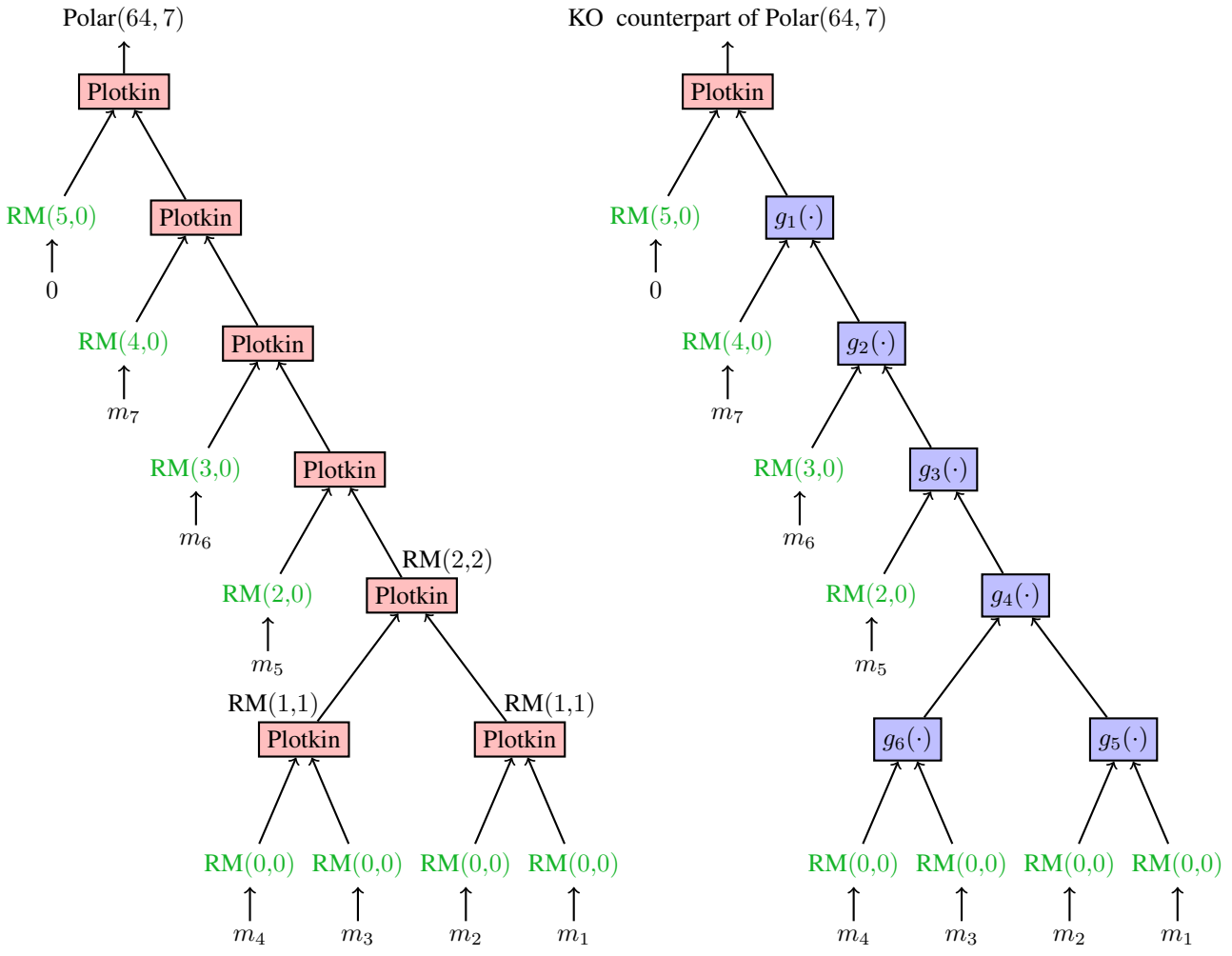
Appendix

A. Polar$(64, 7)$ code

Recall from Section 5 that the Plotkin tree for a Polar code is obtained by freezing a set of leaves in a complete binary tree. These frozen leaves are chosen according to the reliabilities, or equivalently, error probabilities, of their corresponding bit channels. In other words, we first approximate the error probabilities of all the n-bit channels and pick the k-smallest of them using the procedure from (Tal & Vardy, 2013). These k active set of leaves correspond to the transmitted message bits, whereas the remaining $n - k$ frozen leaves always transmit zero.

Here we focus on a specific Polar code: Polar$(64, 7)$, with code dimension $k = 7$ and blocklength $n = 64$. For Polar$(64, 7)$, we obtain these active set of leaves to be $\mathcal{A} = \{48, 56, 60, 61, 62, 63, 64\}$, and the frozen set to be $\mathcal{A}^c = \{1, 2, \cdots 64\} \setminus \mathcal{A}$. Using these set of indices and simplifying the redundant branches, we obtain the Plotkin tree for Polar$(64, 7)$ to be Figure 10. We observe that this Polar Plotkin tree shares some similarities with that of a RM$(6, 1)$ code (with same $k = 7$ and $n = 64$) with key differences at the topmost and bottom most leaves.



(a) Polar$(64, 7)$ encoder

(b) Our KO encoder counterpart

Figure 10. Plotkin trees for the Polar$(64, 7)$ encoder and our neural KO encoder counterpart. Both codes have dimension $k = 7$ and blocklength $n = 64$.

Capitalizing on the encoding tree structure of Polar$(64, 7)$, we build a corresponding KO encoder g_θ which inherits this tree skeleton. In other words, we generalize the Plotkin mapping blocks at the internal nodes of tree, except for the root node, and replace them with a corresponding neural network g_i. Figure 10 depicts the Plotkin tree of our KO encoder.

The KO decoder f_ϕ is designed similarly. Training of the (encoder, decoder) pair (g_θ, f_ϕ) is similar to that of the KO$(8, 2)$ training which we detail in §4.

Figure 11 shows the BLER performance of the Polar$(64, 7)$ code and its competing KO code, for the AWGN channel. Similar to the BER performance analyzed in Figure 9, the KO code is able to significantly improve the BLER performance. For example, we achieve a gain of around 0.5 dB when KO encoder is combined with the MAP decoding. Additionally, the close performance of the KO decoder to that of the MAP decoder confirms its optimality.

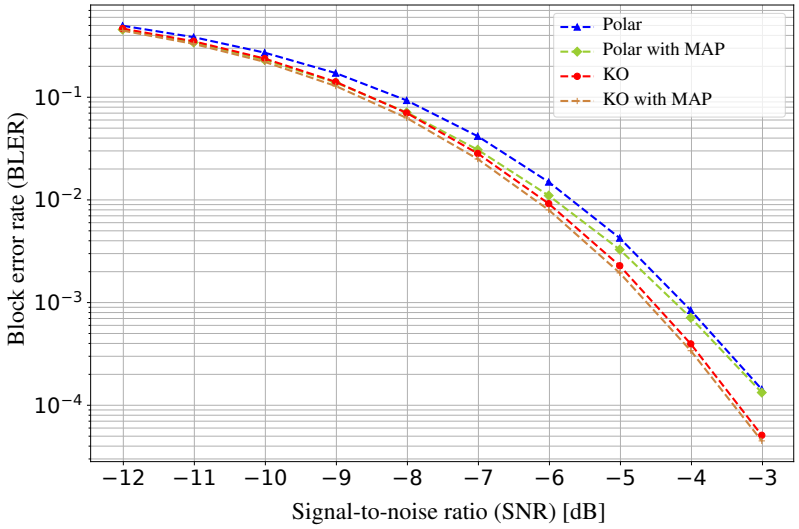


Figure 11. KO code achieves a significant gain over the Polar$(64, 7)$ code in BLER when trained on AWGN channel. KO decoder also matches the optimal MAP decoder.

B. Gains with list decoding

Successive cancellation decoding can be significantly improved by list decoding. List decoding allows one to gracefully tradeoff computational complexity and reliability by maintaining a list (of a fixed size) of candidate codewords during the decoding process. The following figure demonstrates that KO(8,2) code with list decoding enjoys a significant gain over the non-list counterpart. This promising result opens several interesting directions, which are current focuses of active research.

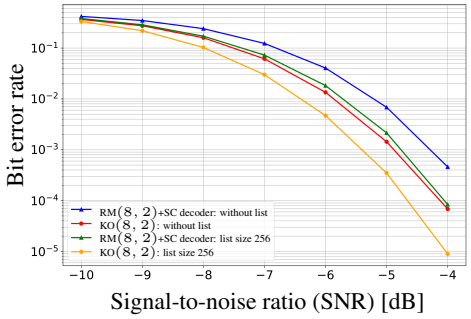


Figure 12. The same KO(8,2) encoder and decoder as those used in Figure 4 achieve a significant gain (without any retraining or fine-tuning) when list decoding is used together with the KO decoder. The magnitude of the gain is comparable to the gain achieved by the same list decoding technique on the successive cancellation decoder of the RM(8,2) code. We used the list decoding from (Dumer & Shabunov, 2006a) but without the permutation technique.

Polar codes with list decoding achieves the state-of-the-art performances (Tal & Vardy, 2015). It is a promising direction to design large block-lengths KO codes (based on the skeleton of Polar codes) that can improve upon the state-of-the-art list-decoded Polar codes. One direction is to train KO codes as we propose and include list decoding after the training. A more ambitious direction is to include list decoding in the training, potentially further improving the performance by discovering an encoder tailored for list decoding.

Unlike Polar codes, RM codes have an extra structure of an algebraic symmetry; a RM codebook is invariant under certain permutations. This can be exploited in list decoding as shown in (Dumer & Shabunov, 2006a), to get a further gain over what is shown in Figure 12. However, when a KO code is trained based on a RM skeleton, this symmetry is lost. A question of interest is whether one can discover nonlinear codes with such symmetry.

C. Tiny KO

As shown in §I.3, each neural component in the KO encoder and decoder has 3 hidden layers with 32 nodes each. For the decoder, the total number of parameters in each decoder neural block is 69×32. We replace all neural blocks with a smaller one with 1 hidden layer of 4 nodes. This decoder neural block has 20 parameters, obtaining a factor of 110 compression in the number of parameters. The computational complexity of this compressed decoder, which we refer to as TinyKO, is within a factor of 4 from Dumer's successive cancellation decoder. Each neural network component has two matrix multiplication steps and one activation function on a vector, which can be fully parallelized on a GPU. With the GPU parallelization, TinyKO has the same time complexity/latency as Dumer's SC decoding.

The following table shows that there is almost no loss in reliability for the compressed KO$(8, 2)$ encoder and decoder in this manner. Training a smaller neural network take about two times more iterations compared to the larger one, although each iteration is faster for the smaller network.

SNR (dB)	TinyKO$(8, 2)$ BER	KO$(8, 2)$ BER
-10	0.38414 $\pm$ 2e-7	0.36555 $\pm$ 2e-7
-9	0.29671 $\pm$ 2e-7	0.27428 $\pm$ 2e-7
-8	0.18037 $\pm$ 2e-7	0.15890 $\pm$ 2e-7
-7	0.07455 $\pm$ 2e-7	0.06167 $\pm$ 1e-7
-6	0.01797 $\pm$ 8e-8	0.01349 $\pm$ 7e-8
-5	2.18083e-3 $\pm$ 3e-8	1.46003e-3 $\pm$ 2e-8
-4	1.18919e-4 $\pm$ 7e-9	0.64702e-4 $\pm$ 4e-9
-3	4.54054e-6 $\pm$ 1e-9	3.16216e-6 $\pm$ 1e-9

Table 2. The smaller TinyKO neural architecture with 100 times smaller number of parameters achieve similar bit-error-rates as the bigger KO architecture.

If one is allowed more computation time (e.g., $O(n^r \log n)$), then (Ye & Abbe, 2020) proposes a recursive projection-aggregation (RPA) decoder for RM(m, r) codes that significantly improves over Dumer's successive cancellation. With list decoding, this is empirically shown to approach the performance of the MAP decoder. It is a promising direction to explore deep learning architectures upon the computation tree of the RPA decoders to design new family of codes.

D. Ablation studies

In comparison to the classical RM codes, the KO codes have two additional features: real-valued codewords and non-linearity. It is thus natural to ask how each of these components contribute to its gains over RM codes. To evaluate their contribution, we did ablation experiments for KO$(8, 2)$: (i) First, we constrain the KO codewords to be binary but allow for non-linearity in the encoder g_θ. The performance of this binarized version KO-b$(8, 2)$ is illustrated in Figure 4. We observe that this binarized KO-b$(8, 2)$ performs similar to RM$(8, 2)$ but worser than KO$(8, 2)$. (ii) We transmit the real-valued codewords but constrain the encoder g_θ to be linear (in real-value operations). At -3 dB the test BER of this linear code is 1.08×10^{-5}, which is significantly worse than KO$(8, 2)$ with 2.97×10^{-6} and similar to RM$(8, 2)$ with 1.30×10^{-5}.

These experiments suggest us that the presence of both the non-linearity and real-valued codewords are necessary for the good performance of KO codes and removal of any of these components hurts its gains.

E. Discussion

E.1. On modulation and practicality of KO codes

We note that our real-valued KO codewords are *entirely practical* in wireless communication; the peak energy of a symbol is only 22.48% larger than the average in KO codes. The impact on the power amplifier is not any different from that of a more traditional modulation (like 16-QAM). Training KO code is a form of *jointly* designing the coding and modulation steps; this approach has a long history in wireless communication (e.g., Trellis coded modulation) but the performance gains have been restricted by the human ingenuity in constructing the heuristics.

E.2. Comparison with LDPC and BCH codes

We expect good performance for BCH at the short blocklengths considered in the paper though with high-complexity (polynomial time) decoders such as ordered statistics decoder (OSD). On the other hand, there does not exist good LDPC codes at the k and n regimes of this paper; thus, we do not expect good performance for LDPC at these regimes.

F. Plotkin construction

Plotkin (1960) proposed this scheme in order to combine two codes of smaller code lengths and construct a larger code with the following properties. It is relatively easy to construct a code with either a high rate but a small distance (such as sending the raw information bits directly) or a large distance but a low rate (such as repeating each bit multiple times). Plotkin construction combines such two codes of rates $\rho_{\boldsymbol{u}} > \rho_{\boldsymbol{v}}$ and distances $d_{\boldsymbol{u}} < d_{\boldsymbol{v}}$, to design a larger block length code satisfying rate $\rho = (\rho_{\boldsymbol{u}} + \rho_{\boldsymbol{v}})/2$ and distance $\min\{2d_{\boldsymbol{u}}, d_{\boldsymbol{v}}\}$. This significantly improves upon a simple time-sharing of those codes, which achieves the same rate but distance only $\min\{d_{\boldsymbol{u}}, d_{\boldsymbol{v}}\}$.

Note: Following the standard convention, we fix the leaves in the Plotkin tree of a first order RM$(m, 1)$ code to be zeroth order RM codes and the full-rate RM$(1, 1)$ code. On the other hand, a second order RM$(m, 2)$ code contains the first order RM codes and the full-rate RM$(2, 2)$ as its leaves.

G. KO$(8, 2)$: Architecture and training

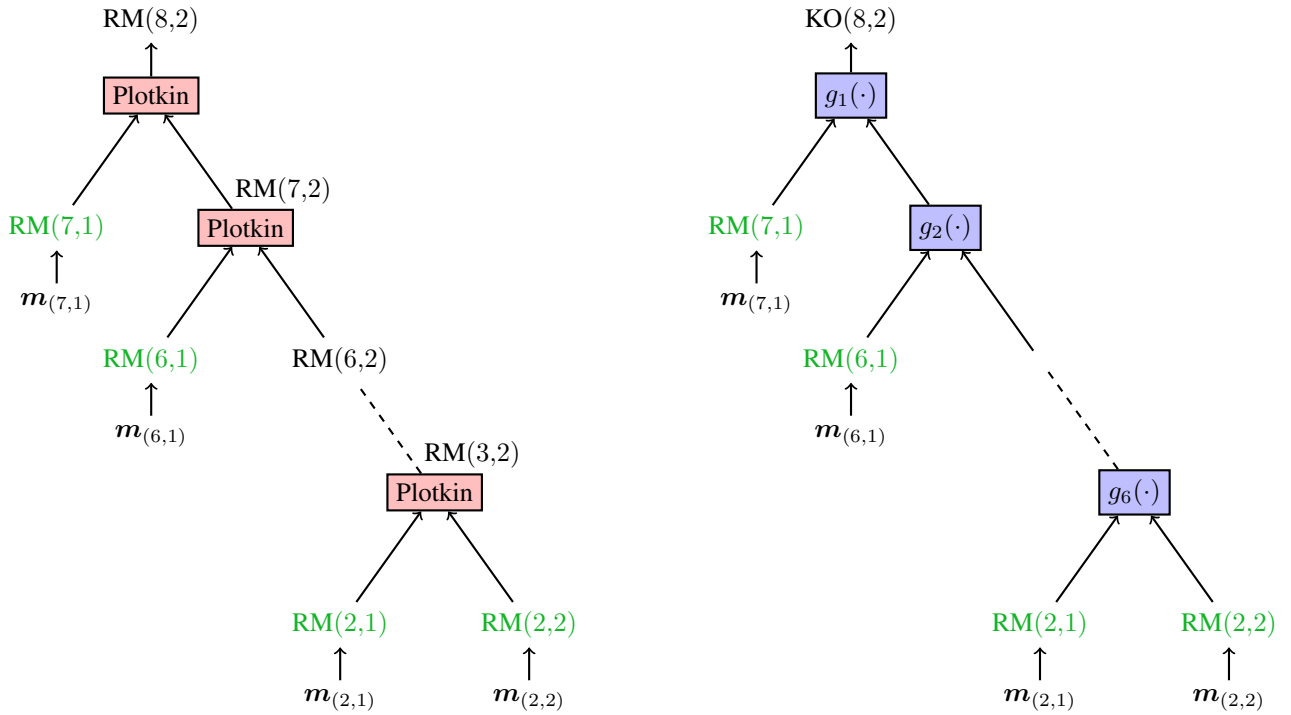
As highlighted in §4, our KO codes improve upon RM codes significantly on a variety of benchmarks. We present the architectures of the KO$(8, 2)$ encoder and the KO$(8, 2)$ decoder, and their joint training methodology that are crucial for this superior performance.

G.1. KO$(8, 2)$ encoder

Architecture. KO$(8, 2)$ encoder inherits the same Plotkin tree structure as that of the second order RM$(8, 2)$ code and thus RM codes of first order and the second order RM$(2, 2)$ code constitute the leaves of this tree, as highlighted in Figure 13b. On the other hand, a critical distinguishing component of our KO$(8, 2)$ encoder is a set of encoding neural networks $g_\theta = \{g_1, \ldots, g_6\}$ that strictly generalize the Plotkin mapping. In other words, we associate a neural network $g_i \in g_\theta$ to each internal node i of this tree. If $\boldsymbol{v}$ and $\boldsymbol{u}$ denote the codewords arriving from left and right branches at this node, we combine them non-linearly via the operation $(\boldsymbol{u}, \boldsymbol{v}) \mapsto g_i(\boldsymbol{u}, \boldsymbol{v})$.

We carefully parametrize each encoding neural network g_i so that they generalize the classical Plotkin map $\mathrm{Plotkin}(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}, \boldsymbol{u} \oplus \boldsymbol{v})$. In particular, we represent them as $g_i(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}, \widetilde{g}_i(\boldsymbol{u}, \boldsymbol{v}) + \boldsymbol{u} \oplus \boldsymbol{v})$, where $\widetilde{g}_i : \mathbb{R}^2 \to \mathbb{R}$ is a neural network of input dimension 2 and output size 1. Here $\widetilde{g}_i$ is applied coordinate-wise on its inputs $\boldsymbol{u}$ and $\boldsymbol{v}$. This clever parametrization can also be viewed as a skip connection on top of the Plotkin map. Similar skip-like ideas have been successfully used in the literature though in a different context of learning decoders (Welling, 2020). On the other hand, we exploit these ideas for both encoders and decoders which further contribute to significant gains over RM codes.

Encoding. From an encoding perspective, recall that the KO$(8, 2)$ code has code dimension $k = 37$ and block length $n = 256$. Suppose we wish to transmit a set of 37 message bits denoted as $\boldsymbol{m} = (\boldsymbol{m}_{(2,2)}, \boldsymbol{m}_{(2,1)}, \ldots, \boldsymbol{m}_{(7,1)})$ through our KO$(8, 2)$ encoder. We first encode the block of four message bits $\boldsymbol{m}_{(2,2)}$ into a RM$(2, 2)$ codeword $\boldsymbol{c}_{(2,2)}$ using its corresponding encoder at the bottom most leaf of the Plotkin tree. Similarly we encode the next three message bits $\boldsymbol{m}_{(2,1)}$ into an RM$(2, 1)$ codeword $\boldsymbol{c}_{(2,1)}$. We combine these codewords using the neural network g_6 at their parent node, which yields the codeword $\boldsymbol{c}_{(3,2)} = g_6(\boldsymbol{c}_{(2,2)}, \boldsymbol{c}_{(2,1)}) \in \mathbb{R}^8$. The codeword $\boldsymbol{c}_{(3,2)}$ is similarly combined with its



(a) RM(8,2) encoder

(b) KO(8,2) encoder

Figure 13. Plotkin trees for RM(8,2) and KO(8,2) encoders. Leaves are highlighted in green. Both codes have dimension $k = 37$ and blocklength $n = 256$.

corresponding left codeword and this procedure is thus recursively carried out till we reach the top most node of the tree, which outputs the codeword $\boldsymbol{c}_{(8,2)} \in \mathbb{R}^{256}$. Finally we obtain the unit-norm KO$(8, 2)$ codeword $\boldsymbol{x}$ by normalizing $\boldsymbol{c}_{(8,2)}$, i.e. $\boldsymbol{x} = \boldsymbol{c}_{(8,2)}/\|\boldsymbol{c}_{(8,2)}\|_2$.

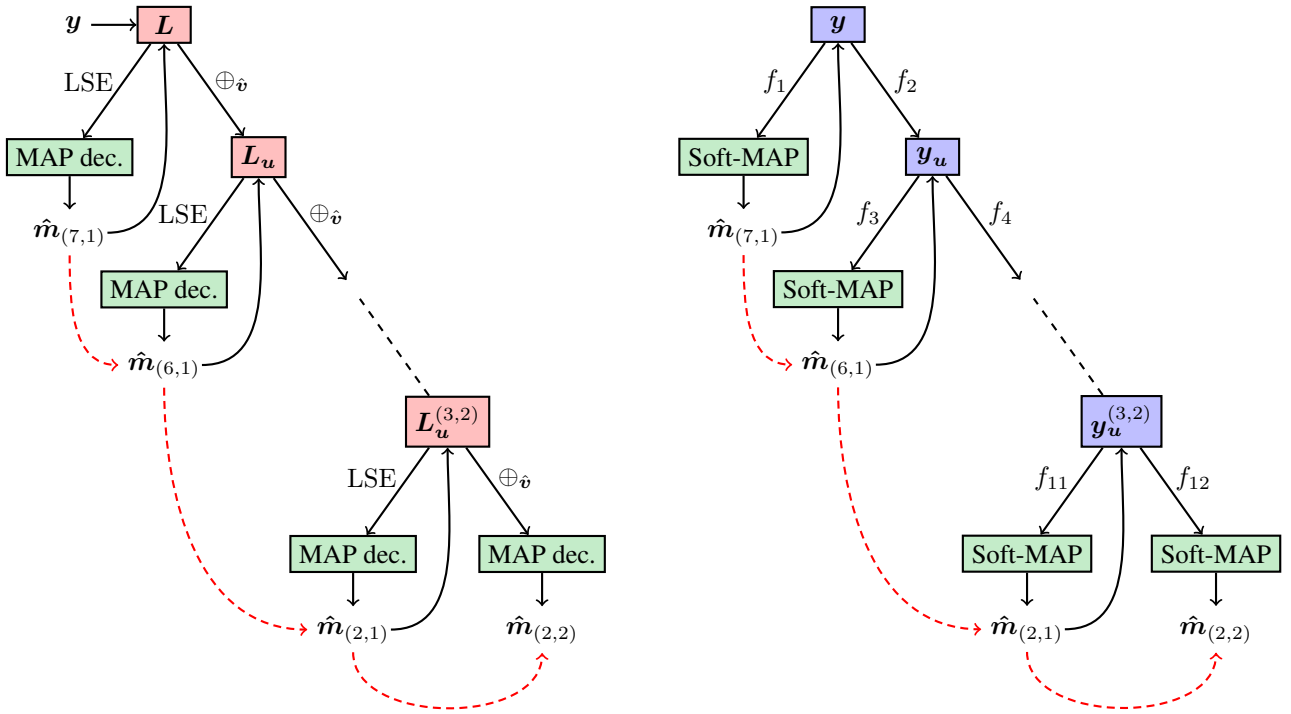
Note that the map of encoding the message bits $\boldsymbol{m}$ into the codeword $\boldsymbol{x}$, i.e. $\boldsymbol{x} = g_\theta(\boldsymbol{m})$, is differentiable with respect to θ since all the underlying operations at each node of the Plotkin tree are differentiable.

G.2. KO$(8, 2)$ decoder

Architecture. Capitalizing on the recursive structure of the encoder, the KO$(8, 2)$ decoder decodes the message bits from top to bottom, similar in style to Dumer's decoding in §2. More specifically, at any internal node of the tree we first decode the message bits along its left branch, which we utilize to decode that of the right branch and this procedure is carried out recursively till all the bits are recovered. At the leaves, we use the Soft-MAP decoder to decode the bits.

Similar to the encoder g_θ, an important aspect of our KO$(8, 2)$ decoder is a set of decoding neural networks $f_\phi = \{f_1, f_2, \ldots, f_{11}, f_{12}\}$. For each node i in the tree, $f_{2i-1} : \mathbb{R}^2 \to \mathbb{R}$ corresponds to its left branch whereas $f_{2i} : \mathbb{R}^4 \to \mathbb{R}$ corresponds to the right branch. The pair of decoding neural networks (f_{2i-1}, f_{2i}) can be viewed as matching decoders for the corresponding encoding network g_i: While g_i encodes the left and right codewords arriving at this node, the outputs of f_{2i-1} and f_{2i} represent appropriate Euclidean feature vectors for decoding them. Further, f_{2i-1} and f_{2i} can also be viewed as a generalization of Dumer's decoding to nonlinear real codewords: f_{2i-1} generalizes the LSE function, while f_{2i} extends the operation $\oplus_{\hat{\boldsymbol{v}}}$. More precisely, we represent $f_{2i-1}(\boldsymbol{y}_1, \boldsymbol{y}_2) = \widetilde{f}_{2i-1}(\boldsymbol{y}_1, \boldsymbol{y}_2) + \text{LSE}(\boldsymbol{y}_1, \boldsymbol{y}_2)$ whereas $f_{2i}(\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_{\boldsymbol{v}}, \hat{\boldsymbol{v}}) = \widetilde{f}_{2i}(\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_{\boldsymbol{v}}, \hat{\boldsymbol{v}}) + \boldsymbol{y}_1 + (-1)^{\hat{\boldsymbol{v}}}\boldsymbol{y}_2$, where $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ are appropriate feature vectors from the parent node, and $\boldsymbol{y}_{\boldsymbol{v}}$ is the feature corresponding to the left-child $\boldsymbol{v}$, and $\hat{\boldsymbol{v}}$ is the decoded left-child codeword. We explain about these feature vectors in more detail below. Note that both the functions $\widetilde{f}_{2i-1}$ and $\widetilde{f}_{2i}$ are also applied coordinate-wise.

Decoding. At the decoder suppose we receive a noisy codeword $\boldsymbol{y} \in \mathbb{R}^{256}$ at the root upon transmission of the actual codeword $\boldsymbol{x} \in \mathbb{R}^{256}$ along the channel. The first step is to obtain the LLR feature for the left RM$(7, 1)$ codeword: we obtain



(a) RM$(8, 2)$ decoder

(b) KO$(8, 2)$ decoder

Figure 14. Plotkin tres for the RM$(8, 2)$ and KO$(8, 2)$ decoders. Red arrows indicate the bit decoding order.

this via the left neural network f_1, i.e. $\boldsymbol{y}_v = f_1(\boldsymbol{y}_{1:128}, \boldsymbol{y}_{129:256}) \in \mathbb{R}^{128}$. Subsequently, the Soft-MAP decoder transforms this feature into an LLR vector for the message bits, i.e. $\boldsymbol{L}_{(7,1)} = \text{Soft-MAP}(f_1(\boldsymbol{y}_{1:128}, \boldsymbol{y}_{129:256}))$. Note that the message bits $\boldsymbol{m}_{(7,1)}$ can be hard decoded directly from the sign of $\boldsymbol{L}_{(7,1)}$. Instead here we use their soft version via the sigmoid function $\sigma(\cdot)$, i.e., $\hat{\boldsymbol{m}}_{(7,1)} = \sigma(\boldsymbol{L}_{(7,1)})$. Thus we obtain the corresponding RM$(7, 1)$ codeword $\hat{\boldsymbol{v}}$ by encoding the message $\hat{\boldsymbol{m}}_{(7,1)}$ via an RM$(7, 1)$ encoder. The next step is to obtain the feature vector for the right child. This is done using the right decoder f_2, i.e. $\boldsymbol{y}_{\boldsymbol{u}} = f_2(\boldsymbol{y}_{1:128}, \boldsymbol{y}_{129:256}, \boldsymbol{y}_{\boldsymbol{v}}, \hat{\boldsymbol{v}})$. Utilizing this right feature $\boldsymbol{y}_{\boldsymbol{u}}$ the decoding procedure is thus recursively carried out till we compute the LLRs for all the remaining message bits $\boldsymbol{m}_{(6,1)}, \ldots, \boldsymbol{m}_{(2,2)}$ at the leaves. Finally we obtain the full LLR vector $\boldsymbol{L} = (\boldsymbol{L}_{(7,1)}, \ldots, \boldsymbol{L}_{(2,2)})$ corresponding to the message bits $\boldsymbol{m}$. A simple sigmoid transformation, $\sigma(\boldsymbol{L})$, further yields the probability of each of these message bits being zero, i.e. $\sigma(\boldsymbol{L}) = \mathbb{P}[\boldsymbol{m} = \boldsymbol{0}]$.

Note that the decoding map $f_\phi : \boldsymbol{y} \mapsto \boldsymbol{L}$ is fully differentiable with respect to ϕ, which further ensures a differentiable loss for training the parameters (θ, ϕ).

G.3. Training

Recall that we have the following flow diagram from encoder till the decoder when we transmit the message bits $\boldsymbol{m}$: $\boldsymbol{m} \xrightarrow{g_\theta} \boldsymbol{x} \xrightarrow{Channel} \boldsymbol{y} \xrightarrow{f_\phi} \boldsymbol{L} \xrightarrow{\sigma(\cdot)} \sigma(\boldsymbol{L})$. In view of this, we define an end-to-end differentiable cross entropy loss function to train the parameters (θ, ϕ), i.e.

$$L(\theta, \phi) = \sum_j m_j \log(1 - \sigma(L_j)) + (1 - m_j) \log \sigma(L_j).$$

Finally we run Algorithm 1 on the loss $L(\theta, \phi)$ to train the parameters (θ, ϕ) via gradient descent.

H. Soft-MAP decoder

As discussed earlier (see also Figure 14), Dumer's decoder for second-order RM codes RM$(m, 2)$ performs MAP decoding at the leaves while our KO decoder applies Soft-MAP decoding at the leaves. The leaves of both RM$(m, 2)$ and KO$(m, 2)$

codes are comprised of order-one RM codes and the RM$(2,2)$ code. In this section, we first briefly state the MAP decoding rule over general binary-input memoryless channels and describe how the MAP rule can be obtained in a more efficient way, with complexity $\mathcal{O}(n \log n)$, for first-order RM codes. We then present the generic Soft-MAP decoding rule and its efficient version for first-order RM codes.

MAP decoding. Given a length-n channel LLR vector $\boldsymbol{l} \in \mathbb{R}^n$ corresponding to the transmission of a given (n, k) code, i.e. code dimension is k and block length is n, with codebook $\mathcal{C}$ over a general binary-input memoryless channel, the MAP decoder picks a codeword $\boldsymbol{c}^*$ according to the following rule (Abbe et al., 2020)

$$\boldsymbol{c}^* = \underset{\boldsymbol{c} \in \mathcal{C}}{\operatorname{argmax}} \ \langle \boldsymbol{l}, 1 - 2\boldsymbol{c} \rangle, \tag{5}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner-product of two vectors. Obviously, the MAP decoder needs to search over all 2^k codewords while each time computing the inner-product of two length-n vectors. Therefore, the MAP decoder has a complexity of $\mathcal{O}(n2^k)$. Thus the MAP decoder can be easily applied to decode small codebooks like an RM$(2,2)$ code, that has blocklength $n = 4$ and a dimension $k = 4$, with complexity $\mathcal{O}(1)$. On the other hand, a naive implementation of the MAP rule for RM$(m, 1)$ codes, that have $2^k = 2^{m+1} = 2n$ codewords, requires $\mathcal{O}(n^2)$ complexity. However, utilizing the special structure of order-1 RM codes, one can apply the fast Hadamard transform (FHT) to implement their MAP decoding in a more efficient way, i.e., with complexity $\mathcal{O}(n \log n)$. The idea behind the FHT implementation is that the standard $n \times n$ Hadamard matrix $\mathbf{H}$ contains half of the the $2n$ codewords of an RM$(m, 1)$ code (in ± 1), and the other half are just $-\mathbf{H}$. Therefore, FHT of the vector $\boldsymbol{l}$, denoted by $\boldsymbol{l}_{\text{WH}}$, lists half of the $2n$ inner-products in (5), and the other half are obtained the as $-\boldsymbol{l}_{\text{WH}}$. Therefore, the FHT version of the MAP decoder for first-order RM codes can be obtained as

$$\boldsymbol{c}^* = (1 - \operatorname{sign}(\boldsymbol{l}_{\text{WH}}(i^*))\boldsymbol{h}_{i^*})/2 \quad \text{s.t.} \quad i^* = \underset{i \in [n]}{\operatorname{argmax}} \ |\boldsymbol{l}_{\text{WH}}(i)|, \tag{6}$$

where $\boldsymbol{l}_{\text{WH}}(i)$ is the i-th element of the vector $\boldsymbol{l}_{\text{WH}}$, and $\boldsymbol{h}_i$ is the i-th row of the matrix $\mathbf{H}$. Given that $\boldsymbol{l}_{\text{WH}}$ can be efficiently computed with $\mathcal{O}(n \log n)$ complexity, the FHT version of the MAP decoder for the first-order RM codes, described in (6), has a complexity of $\mathcal{O}(n \log n)$.

Soft-MAP. Note that the MAP decoder and its FHT version involve $\operatorname{argmax}(\cdot)$ operation which is not differentiable. In order to overcome this issue, we obtain the soft-decision version of the MAP decoder, referred to as Soft-MAP decoder, to come up with differentiable decoding at the leaves (Jamali et al., 2021). The Soft-MAP decoder obtains the soft LLRs instead of hard decoding of the codes at the leaves. Particularly, consider an AWGN channel model as $\boldsymbol{y} = \boldsymbol{s} + \boldsymbol{n}$, where $\boldsymbol{y}$ is the length-n vector of the channel output, $\boldsymbol{s} := 1 - 2\boldsymbol{c}$, $\boldsymbol{c} \in \mathcal{C}$, and $\boldsymbol{n}$ is the vector of the Gaussian noise with mean zero and variance σ^2 per element. The LLR of the i-th information bit u_i is then defined as

$$\boldsymbol{l}_{\text{inf}}(i) := \ln \left(\frac{\Pr(u_i = 0 | \boldsymbol{y})}{\Pr(u_i = 1 | \boldsymbol{y})} \right). \tag{7}$$

By applying the Bayes' rule, the assumption of $\Pr(u_i = 0) = \Pr(u_i = 1)$, the law of total probability, and the distribution of the Gaussian noise, we can write (7) as

$$\boldsymbol{l}_{\text{inf}}(i) = \ln \left(\frac{\sum_{\boldsymbol{s} \in \mathcal{C}_i^0} \exp\left(-||\boldsymbol{y} - \boldsymbol{s}||_2^2 / \sigma^2\right)}{\sum_{\boldsymbol{s} \in \mathcal{C}_i^1} \exp\left(-||\boldsymbol{y} - \boldsymbol{s}||_2^2 / \sigma^2\right)} \right). \tag{8}$$

We can also apply the max-log approximation to approximate (8) as follows.

$$\boldsymbol{l}_{\text{inf}}(i) \approx \frac{1}{\sigma^2} \min_{\boldsymbol{c} \in \mathcal{C}_i^1} ||\boldsymbol{y} - \boldsymbol{s}||_2^2 - \frac{1}{\sigma^2} \min_{\boldsymbol{c} \in \mathcal{C}_i^0} ||\boldsymbol{y} - \boldsymbol{s}||_2^2, \tag{9}$$

where $\mathcal{C}_i^0$ and $\mathcal{C}_i^1$ denote the subsets of codewords that have the i-th information bit u_i equal to zero and one, respectively. Finally, given that the length-n LLR vector of the cahhnel output can be obtained as $\boldsymbol{l} := 2\boldsymbol{y}/\sigma^2$ for the AWGN channels, and assuming that all the codewords $\boldsymbol{s}$'s have the same norm, we obtain a more useful version of the Soft-MAP rule for approximating the LLRs of the information bits as

$$\boldsymbol{l}_{\text{inf}}(i) \approx \max_{\boldsymbol{c} \in \mathcal{C}_i^0} \langle \boldsymbol{l}, 1 - 2\boldsymbol{c} \rangle \ - \ \max_{\boldsymbol{c} \in \mathcal{C}_i^1} \langle \boldsymbol{l}, 1 - 2\boldsymbol{c} \rangle. \tag{10}$$

It is worth mentioning at the end that, similar to the MAP rule, one can compute all the 2^k inner products in $\mathcal{O}(n2^k)$ time complexity, and then obtain the soft LLRs by looking at appropriate indices. As a result, the complexity of the Soft-MAP decoding for decoding RM$(m,1)$ and RM$(2,2)$ codes is $\mathcal{O}(n^2)$ and $\mathcal{O}(1)$ respectively. However, one can apply an approach similar to (6) to obtain a more efficient version of the Soft-MAP decoder, with complexity $\mathcal{O}(n \log n)$, for decoding RM$(m,1)$ codes.

I. Experimental details

We provide our code at https://github.com/deepcomm/KOcodes.

I.1. Training algorithm

Algorithm 1: Training algorithm for KO(8,2)

Input: number of epochs T, number of encoder training steps T_{enc}, number of decoder training steps T_{dec}, encoder training SNR SNR_{enc}, decoder training SNR SNR_{dec}, learning rate for encoder lr_{enc}, learning rate for decoder lr_{dec}

Initialize (θ, ϕ)
for T steps **do**
  for T_{dec} steps **do**
    Generate a minibatch of random message bits $\boldsymbol{m}$
    Simulate AWGN channel with SNR_{dec}
    Fix θ, update ϕ by minimizing $L(\theta, \phi)$ using Adam with learning rate lr_{dec}
  for T_{enc} steps **do**
    Generate a minibatch of random message bits $\boldsymbol{m}$
    Simulate AWGN channel with SNR_{enc}
    Fix ϕ, update θ by minimizing $L(\theta, \phi)$ using Adam with learning rate lr_{enc}

Output: (θ, ϕ)

I.2. Hyper-parameter choices for KO(8,2)

We choose batch size $B = 50000$, encoder training SNR $\text{SNR}_{\text{enc}} = -3dB$, decoder trainint SNR $\text{SNR}_{\text{dec}} = -5dB$, number of epochs $T = 2000$, number of encoder training steps $T_{\text{enc}} = 50$, number of decoder training steps $T_{\text{dec}} = 500$. For Adam optimizer, we choose learning rate for encoder $\text{lr}_{\text{enc}} = 10^{-5}$ and for decoder $\text{lr}_{\text{dec}} = 10^{-4}$.

I.3. Neural network architecture of KO(8,2)

I.3.1. Initialization

We design our (encoder, decoder) neural networks to generalize and build upon the classical (Plotkin map, Dumer's decoder). In particular, as discussed in Section G.1, we parameterize the KO encoder g_θ, as $g_i(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}, \widetilde{g}_i(\boldsymbol{u}, \boldsymbol{v}) + \boldsymbol{u} \oplus \boldsymbol{v})$, where $\widetilde{g} : \mathbb{R}^2 \to \mathbb{R}$ is a fully connected neural network, which we delineate in Section I.3.2. Similarly, for KO decoder, we parametrize it as $f_{2i-1}(\boldsymbol{y}_1, \boldsymbol{y}_2) = \widetilde{f}_{2i-1}(\boldsymbol{y}_1, \boldsymbol{y}_2) + \text{LSE}(\boldsymbol{y}_1, \boldsymbol{y}_2)$ and $f_{2i}(\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_{\boldsymbol{v}}, \hat{\boldsymbol{v}}) = \widetilde{f}_{2i}(\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_{\boldsymbol{v}}, \hat{\boldsymbol{v}}) + \boldsymbol{y}_1 + (-1)^{\hat{\boldsymbol{v}}} \boldsymbol{y}_2$, where $\widetilde{f}_{2i-1} : \mathbb{R}^2 \to \mathbb{R}$ and $\widetilde{f}_{2i} : \mathbb{R}^4 \to \mathbb{R}$ are also fully connected neural networks whose architectures are described in Section I.3.4 and Section I.3.3. If $\widetilde{f} \approx 0$ and $\widetilde{g} \approx 0$, we are able to thus recover the standard RM$(8,2)$ encoder and its corresponding Dumer decoder. By initializing all the weight parameters (θ, ϕ) sampling from $\mathcal{N}(0, 0.02^2)$, we are able to approximately recover the performance RM$(8,2)$ at the beginning of the training which acts as a good initialization for our algorithm.

I.3.2. Architecture of $\widetilde{g}_i$

- Dense(units=2×32)
- SeLU()
- Dense(units=32×32)

- SeLU()
- Dense(units=32×32)
- SeLU()
- Dense(units=32×1)

I.3.3. ARCHITECTURE OF $\widetilde{f}_{2i}$

- Dense(units=4×32)
- SeLU()
- Dense(units=32×32)
- SeLU()
- Dense(units=32×32)
- SeLU()
- Dense(units=32×1)

I.3.4. ARCHITECTURE OF $\widetilde{f}_{2i-1}$

- Dense(units=2×32)
- SeLU()
- Dense(units=32×32)
- SeLU()
- Dense(units=32×32)
- SeLU()
- Dense(units=32×1)

J. Results for Order-1 codes

Here we focus on first order KO$(m, 1)$ codes, and in particular KO$(6, 1)$ code that has code dimension k =7 and blocklength $n = 64$. The training of the (encoder, decoder) pair (g_θ, f_ϕ) for KO$(6, 1)$is almost identical to that of the second order RM$(8, 2)$ described in §3. The only difference is that we now use the Plotkin tree structure of the corresponding RM$(6, 1)$ code. In addition, we also train our neural encoder g_θ together with the differentiable MAP decoder, i.e. the Soft-MAP, to compare its performance to that of the RM codes. Figure 15 illustrates these results.

The left panel of Figure 15 highlights that KO$(6, 1)$ obtains significant gain over RM$(6, 1)$ code (with Dumer decoder) when both the neural encoder and decoder are trained jointly. On the other hand, in the right panel, we notice that we match the performance of that of the RM$(6, 1)$ code (with the MAP decoder) when we just train the encoder g_θ (with the MAP decoder). In other words, under the optimal MAP decoding, KO$(6, 1)$ and RM$(6, 1)$ codes behave the same. Note that the only caveat for KO$(6, 1)$ in the second setting is that its MAP decoding complexity is $O(n^2)$ while that of the RM is $O(n \log n)$.

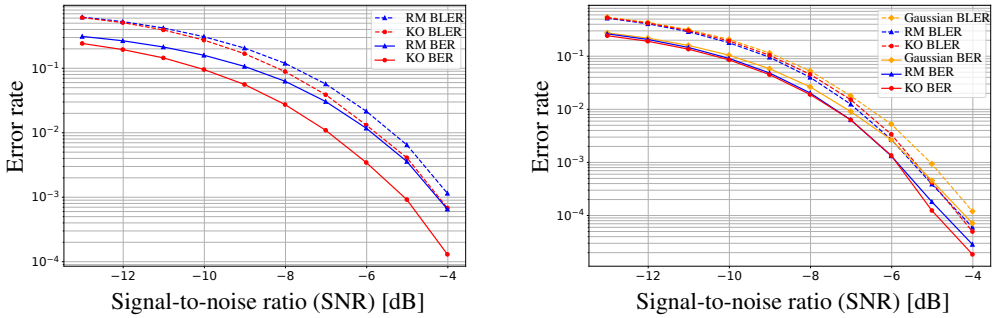


Figure 15. KO$(6, 1)$ code. **Left**: KO$(6, 1)$ code achieves significant gain over RM$(6, 1)$ code (with Dumer) when trained on AWGN channel. **Right**: Under the optimal MAP decoding, KO$(6, 1)$ and RM$(6, 1)$ codes achieve the same performance. Error rates for a random Gaussian codebook are also plotted as a baseline.