Reed-Muller Subcodes: Machine Learning-Aided Design of Efficient Soft Recursive Decoding

Mohammad Vahid Jamali  
EECS Department  
University of Michigan  
mvjamali@umich.edu

Xiyang Liu  
CSE Department  
University of Washington  
xiyangl@cs.washington.edu

Ashok Vardhan Makkuva  
ECE Department  
University of Illinois at Urbana-Champaign  
makkuva2@illinois.edu

Hessam Mahdavifar  
EECS Department  
University of Michigan  
hessam@umich.edu

Sewoong Oh  
CSE Department  
University of Washington  
sewoong@cs.washington.edu

Pramod Viswanath  
ECE Department  
University of Illinois at Urbana-Champaign  
pramodv@illinois.edu

Abstract—Reed-Muller (RM) codes are conjectured to achieve the capacity of any binary-input memoryless symmetric (BMS) channel, and are observed to have a comparable performance to that of random codes in terms of scaling laws. On the negative side, RM codes lack efficient decoders with performance close to that of a maximum likelihood decoder for general parameters. Also, they only admit certain discrete sets of rates. In this paper, we focus on subcodes of RM codes with flexible rates that can take any code dimension from 1 to \( n \), where \( n \) is the blocklength. We first extend the recursive projection-aggregation (RPA) algorithm proposed recently by Ye and Abbe for decoding RM codes. To lower the complexity of our decoding algorithm, referred to as subRPA in this paper, we investigate different ways for pruning the projections. We then derive the soft-decision based version of our algorithm, called soft-subRPA, that is shown to improve upon the performance of subRPA. Furthermore, it enables training a machine learning (ML) model to search for good sets of projections in the sense of minimizing the decoding error rate. Training our ML model enables achieving very close to the performance of full-projection decoding with a significantly reduced number of projections. For instance, our simulation results on a \((64,14)\) RM subcode show almost identical performance for full-projection decoding and pruned-projection decoding with 15 projections picked via training our ML model. This is equivalent to lowering the complexity by a factor of more than 4 without sacrificing the decoding performance.

I. INTRODUCTION

Reed-Muller (RM) codes are among the oldest families of error-correcting codes, and their origin backs to almost seven decades ago [1], [2]. They have received significant renewed interest after the breakthrough invention of polar codes [3], given the close connection between the two classes of codes. The generator matrices for both RM and polar codes are obtained from the same square matrices, the Kronecker powers of a \( 2 \times 2 \) matrix, though by different rules for selecting rows. In fact, such a selection of rows for polar codes is channel-specific but the RM encoder picks the rows with the largest Hamming weights. Therefore, RM codes have a universal construction. Additionally, RM codes provably achieve the Shannon capacity of binary erasure channels (BECs) at any constant rate [4], and that of binary symmetric channels (BSCs) at extremal rates, i.e., at rates converging to zero or one [5]. The long-time belief that RM codes achieve the Shannon capacity over any binary-input memoryless symmetric (BMS) channel, however, still remains an open problem [6]. RM codes are also conjectured to have characteristics similar to those of random codes in terms of both weight enumeration [7] and scaling laws [8].

Despite their excellent performance with maximum likelihood decoders, RM codes still suffer from the lack of an efficient decoding algorithm for general parameters. Among the earlier works on decoding RM codes [1], [9]–[14], Dumer’s recursive list decoding algorithm [9]–[11] provides a trade-off between the decoding complexity and error probability. In other words, it is capable of achieving close to maximum likelihood decoding performance for large enough, e.g., exponential in blocklength, list sizes. Recently, Ye and Abbe [15] proposed a recursive projection-aggregation (RPA) algorithm for decoding RM codes. The RPA algorithm first projects the received corrupted codeword on its cosets. It then recursively decodes the projected codes to, finally, construct the decoded codeword by properly aggregating them. Very recently, building upon the projection pruning idea in [15], a method for reducing the complexity of the RPA algorithm has been explored in [16].

Besides lacking an efficient decoder in general, the structure of RM codes does not allow choosing a flexible rate. To clarify this, let \( k \) and \( n \) denote the code dimension and blocklength, respectively. Due to the underlying Kronecker product structure of RM codes, the code blocklength is a power of two, i.e., \( n = 2^m \), where \( m \) is a design parameter. Additionally, RM codes possess another parameter \( r \), that stands for the order of the code, where \( 0 \leq r \leq m \). Then, given the code blocklength \( n \), one can only construct RM codes with \( m+1 \) possible values for the code rate, each corresponding to a given code order \( r \).

This research is inspired by the aforementioned two critical issues of RM codes. More specifically, we target subcodes of RM codes, and our primary goal is to come up with low-complexity decoders for the RM subcodes. To this end, we first extend the RPA algorithm to what we call “subRPA” in this paper. Similar to the RPA algorithm, subRPA starts by projecting the received corrupted codeword onto the cosets. However, since the projected codes are no longer RM codes
of lower orders, their corresponding generator matrices have different ranks (i.e., different code dimensions). SubRPA applies the optimal maximum a posteriori (MAP) decoder at the bottom layer given the low dimension of the projected codes at that layer. It then aggregates the reconstructions to recursively decode the received codeword. Next, we derive the soft-decision based version of our algorithm, called “soft-subRPA”, that improves upon the performance of subRPA. We further investigate various ways for pruning the projections to reduce the complexity of the proposed algorithms with negligible performance loss. Enabled by our soft-subRPA algorithm, we train a machine learning (ML) model to search for projections, in efficiently decoding RM subcodes. In this case of general binary-input memoryless channels) into the log-likelihood ratio (LLR) vector of the channel output (in the case of BSC) or the received corrupted binary vector (in the case of $B=0$), the quotient space $\mathcal{E}:=\mathbb{F}_2^\tau$. Moreover, each codeword $c = (c(z), z \in \mathcal{E}) \in \mathcal{R}(m, r)$, that is also indexed by the binary vectors $z$, is defined as the evaluations of an $m$-variate polynomial with degree at most $r$ at points $z \in \mathcal{E}$.

### B. RP A Decoding Algorithm

The RPA algorithm is comprised of the following three building blocks/operations.

1) **Projection:** The RPA algorithm starts by projecting the received corrupted binary vector (in the case of BSC) or the log-likelihood ratio (LLR) vector of the channel output (in the case of general binary-input memoryless channels) into the subspaces of $\mathcal{E}$. Considering $B$ as a $s$-dimensional subspace of $\mathcal{E}$, with $s \leq r$, the quotient space $\mathcal{E}/B$ contains all the cosets of $B$ in $\mathcal{E}$. Each coset $\tau$ has the form $\tau = z + B$ for some $z \in \mathcal{E}$. Then, in the case of BSC, the projection of the channel binary output $y = (y(z), z \in \mathcal{E})$ on the cosets of $B$ is defined as

$$
y_B := \left\{ y_B(\tau) \in \mathcal{E}/B \right\}, \text{s.t. } y_B(\tau) := \bigoplus_{z \in \tau} y(z), \quad (3)$$

where $\bigoplus$ denotes the coordinate-wise addition in $\mathbb{F}_2$. For the binary-input memoryless channels the RPA algorithm works on the projection of the channel output LLR vector $I$. In the case of a one-dimensional subspace $B$, the projected LLR vector can be obtained as $I_B := (I_B(\tau), \tau \in \mathcal{E}/B)$, where

$$
I_B(\tau) := \ln \left( \exp \left( \sum_{z \in \tau} I(z) \right) + 1 \right) - \ln \left( \sum_{z \in \tau} \exp(I(z)) \right). \quad (4)
$$
2) Decoding the Projected Outputs: Once the decoder projects the channel output (y or l), it starts recursively decoding the projected outputs, i.e., it projects them into new subspaces and continues until the projected outputs correspond to order-1 RM codes. The decoder then applies the fast Hadamard transform (FHT) \([17]\) to efficiently decode order-1 codes. Using the FHT algorithm, one can implement the MAP decoder for the first-order RM codes with complexity \(\mathcal{O}(n \log n)\) instead of \(\mathcal{O}(n^2)\). Once the first-order codes are decoded, the algorithm aggregates the outputs (as explained next) to decode the codes at a higher layer. The decoder may also iterate the whole process, at each middle decoding step, several times to ensure the convergence of the algorithm.

3) Aggregation: At each layer in the decoding process (and each point/node in the decoding tree), the decoder needs to aggregate the output of the channel at that point with the decoding results of the next (underneath) layer to update the channel output. Note that the channel output at a given point can be either the actual channel output (y or l) or the projected ones, depending on the position of that point in the decoding tree of the recursive algorithm. Several aggregation algorithms are presented in \([15]\) for one- and two-dimensional subspaces. We refer the reader to \([15]\) for more details on the aggregation methods.

III. EFFICIENT DECODING OF RM SUBCODES

Let \(F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\), and define \(P_{n \times n} = F^\otimes m\), i.e., the \(m\)-th Kronecker power of \(F\). It can be observed that the encoding of \(\mathcal{RM}(m, r)\) (described in Section II-A) can be equivalently obtained by choosing the rows of the square matrix \(P_{n \times n}\) that have a Hamming weight of at least \(2^{m-r}\). The resulting generator matrix \(G_{k \times n}\) then has exactly \(\binom{m}{r}\) rows with the Hamming weight \(n/2^r\), for \(0 \leq r \leq m\).

Note that the RM encoder does not allow choosing any desired code dimension; it should be of the form \(k = \sum_{i=0}^{m-1} \binom{m}{r}\) for some \(r \in \{0, 1, \ldots, m\}\). Suppose that we want to construct a subcode of \(\mathcal{RM}(m, r)\) with a dimension \(k\) such that \(k_1 < k < k_u\), where \(k_1 := \sum_{i=0}^{r-1} \binom{m}{r}\), \(r \in [m]\), and \(k_u := \sum_{i=0}^{m} \binom{m}{r}\). Given that the construction of RM codes corresponds to picking rows of \(P_{n \times n}\) that have the highest Hamming weights, the first \(k_1\) rows of the generator matrix \(G_{k \times n}\) will be the same as the generator matrix of the lower rate RM code, i.e., \(\mathcal{RM}(m, r-1)\), that has a Hamming weight of at least \(2^{m-r+1}\). It then remains to pick extra \(k - k_1\) rows from \(P_{n \times n}\). These will be picked from the additional \(k_u - k_1 = \binom{m}{r}\) rows in \(\mathcal{RM}(m, r)\) since they all have the same Hamming weight of \(2^{m-r}\) which is the next largest Hamming weight. In a sense, we limit our attention to RM subcodes that, roughly speaking, sit between two RM codes of consecutive orders. More specifically, they are subcodes of \(\mathcal{RM}(m, r)\) and also contain \(\mathcal{RM}(m, r-1)\) as a subcode, for some \(r \in [m]\).

The question is then how to choose the extra \(k - k_1\) rows out of these \(\binom{m}{r}\) rows of weight \(2^{m-r}\) to construct an RM subcode of dimension \(k\) as specified above? This is a very important question requiring a separate follow-up work and is beyond the scope of this paper. In the meantime, we provide some insights regarding the encoding of RM subcodes in Section III-C after describing our decoding algorithms in Sections III-A and III-B with respect to a generic generator matrix \(G_{k \times n}\). Our results show that randomly selecting a subset of these rows is not always good. Indeed, some selections are better than the others, and also the set of good rows can depend on the decoding algorithm.

A. SubRPA Decoding Algorithm

Before we delve into the description of our decoding algorithms, we first need to emphasize some important facts. Remark 1. The result of the projection operation corresponds to a code with the generator matrix that is formed by merging (i.e., binary addition of) the columns of the original code generator matrix indexed by the cosets of the projection subspace. This is clear for the BSC model, as formulated in \([3]\). Additionally, for general BMS channels, the objective is to estimate the projected codewords \(c_{\mathcal{E}}(\tau)\)’s, \(\tau \in \mathbb{E}/\mathcal{B}\), based on the channel (projected) LLRs \([15]\); hence, the same principle follows for any BMS channels.

Proposition 1. Let \(\mathcal{C}\) be a subcode of \(\mathcal{RM}(m, r)\) with dimension \(k\) such that \(k_1 < k < k_u\), where \(k_1 := \sum_{i=0}^{r-1} \binom{m}{i}\), \(r \in [m]\), and \(k_u := \sum_{i=0}^{m} \binom{m}{i}\). The projection of this code into \(s\)-dimensional subspaces of \(\mathbb{E}\), \(1 \leq s \leq r - 1\), results in subcodes of \(\mathcal{RM}(m-s, r-s)\) or \(\mathcal{RM}(m-s, r-1-s)\) codes.

Proof: The projection of \(\mathcal{RM}(m, r)\) into \(s\)-dimensional subspaces, \(1 \leq s \leq r\) is an \(\mathcal{RM}(m-s, r-s)\) code \([15]\). The code \(\mathcal{C}\) is constructed by removing \(k_u - k\) rows of the generator matrix of \(\mathcal{RM}(m, r)\) that are not in the generator matrix of \(\mathcal{RM}(m, r-1)\) while the projection of \(\mathcal{RM}(m, r-1)\) into \(s\)-dimensional subspaces, \(1 \leq s \leq r-1\), is an \(\mathcal{RM}(m-s, r-1-s)\) code. Now, given that each \(s\)-dimensional projection is equivalent to partitioning \(n\) columns of the generator matrix into \(n/2^s\) groups of \(2^s\) columns and adding them in the binary field (see Remark 1), the generator matrices of the projected codes contain rows of the generator matrix of \(\mathcal{RM}(m, r-1)\) and, possibly, a subset of the rows of the generator matrix of \(\mathcal{RM}(m-s, r-s)\) that are not in the generator matrix of \(\mathcal{RM}(m-s, r-1-s)\). More precisely, if the selected additional \(k - k_1\) rows do not contribute in the rank of the merged matrix according to a given subspace, the projected code into that subspace is an \(\mathcal{RM}(m-s, r-1-s)\) code. On the other hand, if the removed \(k_u - k\) rows do not contribute in that rank, the projected code is an \(\mathcal{RM}(m-s, r-s)\) code. Otherwise, that projected code is a subcode of \(\mathcal{RM}(m-s, r-s)\).

Hereinafter, for the sake of brevity, we simply say that “the projections of a subcode of \(\mathcal{RM}(m, r)\) code into the \(s\)-dimensional subspaces of \(\mathbb{E}\) are subcodes of \(\mathcal{RM}(m-s, r-s)\);” however, we still mean the precise statement of Proposition 1. Now, we are ready to present our decoding algorithms for RM subcodes. Our algorithms are based on one-dimensional (1-D) subspaces. However, they can be easily generalized to the case of \(s\)-dimensional subspaces.

The subRPA algorithm is very similar to the RPA algorithm. More precisely, it first projects the code \(\mathcal{C}\), that is a
subcode of $\mathcal{R}M(m, r)$, into 1-D subspaces to get subcodes of $\mathcal{R}M(m - 1, r - 1)$ at the next layer. It then recursively applies the subRPA algorithm to decode these projected codes. Next, it aggregates the decoding results of the next layer with the output LLRs of the current layer (similar to [15] Algorithm 4) to update the LLRs. Finally, it iterates this process several times to ensure the convergence of the algorithm, and takes the sign of the updated LLRs to obtain the decoded codewords.

The main distinction between subRPA algorithm and RPA algorithm, however, is the decoding of the projected codes at the bottom layer. Based on Proposition 1 after $r - 1$ layers of 1-D projections, the decoder ends up with subcodes of $\mathcal{R}M(m - r + 1, 1)$ at the bottom layer. These projected codes can have different dimensions though all are less than or equal to $m - r + 2$. Therefore, the subRPA algorithm, manageably, applies the MAP decoding at the bottom layer. Given that the projected codewords at the bottom layer are not all from the same codes, the MAP decoding should be carefully performed. Based on Remark 1, the projected codes at the bottom layer can be obtained from the so-called projected generator matrices of dimension $k \times 2^{m-r+1}$, after $r - 1$ times (binary) merging of the $2^m$ columns of the original generator matrix $G_{k \times n}$. However, many of these $k$ rows of the projected generator matrices are linearly dependent. In fact, all of these matrices have ranks (i.e., code dimensions) of less than or equal to $m - r + 2$. In order to facilitate the MAP decoding at the bottom layer, we can pre-compute and store the codebook of each projected code at the bottom layer. Particularly, let $R_t$ be the rank of the $t$-th projected generator matrix $G_p(t)$ at the bottom layer, $t \in [T]$, where $T$ is the total number of projected codes at the bottom layer (that depends on the number of layers as well as the number of projections per layer). Now, we can pre-compute the codebook $C_p(t)$ that contains the $2^{R_t} \times (n/2^{r-1})$ codewords $c_{p,i}^{(t)}$, $i \in [2^{R_t}]$, of the $t$-th projected code at the bottom layer. Now, given the projected LLR vector $l_p(t)$ of length $n/2^{r-1}$ at the bottom layer, we pick the codeword $c_{p,i}^{(t)}$, that maximizes the MAP rule for BMS channels [15], i.e.,

$$y_t = c_{p,i}^{(t)}, \quad \text{s.t.} \quad i^* = \arg\max_{i \in [2^{R_t}]} \langle l_p(t), 1 - 2c_{p,i}^{(t)} \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the inner (dot) product of two vectors.

**B. Soft-SubRPA Algorithm**

In this section, we derive the soft-decision version of the subRPA algorithm, referred to as “soft-subRPA” in this paper. The soft-subRPA algorithm obtains soft decisions at the bottom layers instead of performing hard MAP decodings; this process is called “soft-MAP” in this paper. Additionally, the decoder applies a different rule to aggregate the soft decisions obtained from the next layers with the LLRs available at the current layer; we refer to this aggregation process as “soft-aggregation”. The soft-subRPA algorithm not only improves upon the performance of the subRPA but also replaces the hard MAP decodings at the bottom layer with a differentiable operation that, in turn, enables training an ML model as delineated in Section III-E.

| Algorithm 1 Soft-MAP Algorithm for AWGN Channels |
|---|
| **Input:** The LLR vector $l_p$; the generator matrix $G_p$; the codebook $C_p$; and the matrix $U$ of the information sequences. |
| **Output:** Soft decisions (i.e., the updated LLR vector) $l$. |

1: Set $k$ equal to the number of rows in $G_p$.
2: Initialize $l_{\inf}$ as an all-zero vector of length $k$.
3: $\hat{C} \leftarrow 1 - 2C \triangleright C$ is the codebook matrix (in binary).
4: $\tilde{l} \leftarrow l_p \hat{C}^T \triangleright$ matrix mul. of $l_p$, with the transpose of $\hat{C}$.
5: for $i = 1, 2, \ldots, k$ do inf.
5.1: if $U(\cdot, i)$ (the $i$-th column) is not fixed to 0 or 1 then
5.2: $l_{\inf}(i) \leftarrow \max_{i'} I(i') - \max_{i'} I(i')$
6: end if
7: end for

8: Initialize $l_{\enc}$ as an all-zero vector of length $n'$.
9: $L \leftarrow \text{repeat}(l_{\inf}^T, 1, n') \triangleright$ make $n'$ copies of $l_{\inf}$.
10: $V = L \odot G_p \triangleright$ element-wise matrix multiplication.
11: for $j = 1, 2, \ldots, n'$ do
12: $v \leftarrow \text{vector containing nonzero elements of } V(\cdot, j)$.
13: $l_{\enc}(j) \leftarrow \prod_i \text{sign}(v(j')) \times \text{min}_{j'} |v(j')|$
14: end for
15: $l \leftarrow l_{\enc}$.
16: return $l$.

The soft-MAP algorithm for making soft decisions on the projected codes at the bottom layer, that are subcodes of first-order RM codes, is presented in Algorithm 1 for the case of additive white Gaussian noise (AWGN) channels. The process is comprised of two main steps: 1) obtaining the LLRs of the information bits, and 2) obtaining the soft decisions (i.e., LLRs) of the coded bits using that of information bits. Note that we invoke max-log and min-sum approximations, to be clarified later, in Algorithm 1. For the sake of brevity, let us drop the superscript $t$. Particularly, let $R$ be the rank of the projected generator matrix $G_p$ of a projected code at the bottom layer with codebook $C_p$. Also, assume a $2^R \times k$ matrix $U$ that lists all $2^R$ length-$k$ sequences of bits that produce the codebook $C_p$ (through modulo-2 matrix multiplication $UG_p$). Note that only $R$ indices of these length-$k$ sequences contain the information bits and the remaining indices are always fixed to either 0 or 1. The objective of the first step is to obtain the LLRs of these $R$ information bits using the available projected LLR vector $l_p$. This can be done, using (9) in Appendix A invoking max-log approximation, as described in Algorithm 1. Note that the LLRs of the $k - R$ indices that do not contain the information bits are set to zero.

Once we have the LLRs of the information bits, we combine them according to the columns of $G_p$ to obtain the LLRs of the encoded bits $l_{\enc}$. The codewords in $C_p$ are obtained by the multiplication of $UG_p$, i.e., each $j$-th coded bit, $j \in [n']$, where $n'$ is the code length, is obtained based on the linear combination of the information bits $u_j$, according to the $j$-th column of $G_p$. Therefore, we can apply the well-known min-sum approximation to calculate the LLR vector of
the coded bits as \( l_{\text{enc}} := (l_{\text{enc}}(j), j \in \mathbb{N}') \), where
\[
l_{\text{enc}}(j) = \prod_{i \in \Delta_j} \text{sign}(l_i) \times \min_{i \in \Delta_j} |l_i|, \tag{6}
\]
where \( \Delta_j \) is the set of indices defining the nonzero elements in the \( j \)-th column of \( G_p \). This process is summarized in Algorithm 1 in an efficient way. The decoder may also iterate the whole process several times to assure the convergence of the soft-MAP algorithm.

Finally, given the soft decisions at the bottom layer, the decoder needs to aggregate the decisions with the current LLRs. In the following, we first define the “soft-aggregation” scheme as an extension of the aggregation method in [5, Algorithm 4] for the case of soft decisions.

**Definition 1** (Soft-Aggregation). Let \( l \) be the vector of the channel LLRs, with length \( n = 2^m \), at a given layer. Suppose that there are \( Q \) 1-D subspaces \( \mathbb{B}_q \), \( q \in \{0, 1\} \), to project this LLR vector at the next layer (in the case of full-projection decoding, there are \( n - 1 \) 1-D subspaces, hence \( Q = n - 1 \)). Also, let \( \hat{l}_q \) denote the length-\( n/2 \) vector of soft decisions of the projected LLRs according to Algorithm 7. The “soft-aggregation” of \( l \) and \( \hat{l}_q \)’s is defined as a length-\( n \) vector \( l' := (l'(z), z \in \mathbb{F}_2^n) \) where
\[
l'(z) = \frac{1}{Q} \sum_{q=1}^{Q} \text{tanh} \left( \frac{\hat{l}_q([z + \mathbb{B}_q])/2}{2} \right) l(z + z_q), \tag{7}
\]
where \( z_q \) is the nonzero vector of the 1-D subspace \( \mathbb{B}_q \), and \( [z + \mathbb{B}_q] \) is the coset containing \( z \) for the projection into \( \mathbb{B}_q \).

In order to observe (7), recall that the objective of the aggregation step is to update the length-\( n \) channel LLR vector \( l \) to \( l' \) given the soft decisions of the projected codes. \( \hat{l}_q([z + \mathbb{B}_q]) \) serves as a soft estimate of the binary addition of the coded bits at positions \( z \) and \( z + z_q \). Hence, following the same arguments as [5], if that combined bit is 0, then the updated LLR at position \( z \) should take the same sign as the channel LLR at position \( z + z_q \). Note that this happens with probability \( a_0 := 1 / |1 + \exp(-\hat{l}_q([z + \mathbb{B}_q]))| \). Similarly, with probability \( a_1 := 1 / |1 + \exp(\hat{l}_q([z + \mathbb{B}_q]))| \) the combined bit is 1, and hence the updated LLR at position \( z \) and \( l(z + z_q) \) should have different signs. Therefore, given a projection subspace \( \mathbb{B}_q \), one can update the channel LLR as \( a_0 \times l(z + z_q) + a_1 \times -l(z + z_q) \). Taking the average over all \( Q \) projections then yields the soft-aggregation rule in (7).

It is worth mentioning that one can also apply the following equation to update the channel LLR as
\[
l'(z) = \frac{1}{Q} \sum_{q=1}^{Q} \ln \left( \frac{1 + e^{\hat{l}_q([z + \mathbb{B}_q]) + l(z + z_q)}}{e^{\hat{l}_q([z + \mathbb{B}_q])} + e^{l(z + z_q)}} \right). \tag{8}
\]

The rationale behind (8) follows similar arguments as above and then deriving the LLR of the sum of two binary random variables given the LLRs of each of them. Therefore, (8) is an exact expression assuming independence among the involved LLR components. Our empirical observations, however, suggest almost identical results for either aggregation methods. Therefore, given the complexity of computing expressions like (8), one can reliably apply our proposed soft-aggregation method in Definition 1.

### C. Encoding Insights

The main objective of this paper is to develop schemes for decoding RM subcodes with low complexity. In this subsection, we provide some insights on how the design of the encoder can affect the decoding complexity as well as the performance. First, in order to further highlight the efficiency of RM subcodes, in Figure 2 we compare the block error rate (BLER) performance of RM subcodes with time-sharing (TS) between RM codes given the optimal MAP decoding. We consider two RM subcodes with parameters \((n, k) = (64, 14)\) and \((64, 18)\). The generator matrix construction for these codes is based on having the largest ranks for the projected generator matrices which will be clarified at the end of this subsection. The TS performance is obtained assuming that the transmitter employs an \(\mathcal{R} \mathcal{M}(6, 2)\) encoder in a portion of the time and an \(\mathcal{R} \mathcal{M}(6, 1)\) encoder in the remaining \((1 - \alpha)\) portion, where \(\alpha = 7/15\) and 11/15, to achieve the same code rates 14/64 and 18/64, respectively. It is observed that the RM subcodes with the rates 14/64 and 18/64 achieve more than 1 dB and 4 dB gains, respectively, compared to the TS counterparts. Also, the performance of the RM subcode with rate 18/64 is almost 0.2 dB better than the performance of the lower rate code with TS. Note that all the simulation results in this paper are obtained from more than 10^4 trials of random codewords (except \(\mathcal{R} \mathcal{M}(6, 2)\) with MAP decoding that has 10^4 trails). Additionally, throughout the paper, we define the signal-to-noise ratio (SNR) as \(\text{SNR} := 1/(2\sigma^2)\) and the energy-per-bit \(E_b\) to the noise ratio as \(E_b/N_0 := n/(2\kappa\sigma^2)\), where \(\sigma^2\) is the noise variance.

As discussed earlier, our decoding algorithms perform MAP or soft-MAP decoding at the bottom layer. Also, the dimension of the projected codes for the bottom layer (i.e., the rank of the projected generator matrices) can be different. This is in contrast to RM codes that always result in the same dimension for the projected codes at the bottom layer. Therefore, an immediate approach for encoding RM subcodes to achieve a
lower decoding complexity is to construct the code generator matrix such that the projected codes at the bottom layer have smaller dimensions, and thus the decodings at the bottom layer have lower complexities. In other words, let \( L := \sum_{l=1}^{T} 2^R_l \) represent a rough evaluation of the decoding complexity at the bottom layer, i.e., the decoding complexity at the bottom layer is roughly a constant times \( L \). Then, among all \( \binom{k_t}{k} \) possible selections of the generator matrix \( G_{k \times n} \), we can choose the ones that achieve a smaller \( L \). This encoding scheme leads to reduction in the decoding complexity of our algorithms but it also affects the performance. In order to investigate the effect of this methodology, in Figure 3 we consider four different selections of the generator matrix for the \((64,14)\) RM subcode. In particular, \( G_{\max} \) and \( G_{\max2} \) have the largest and the second largest values of \( L = 2568 \) and \( 2532 \), respectively. Also, \( G_{\min} \) has the minimum value of \( L = 1482 \). And, \( G_{\min,15} \) has the minimum value of \( \sum L = 108 \) on 15 projections but a relatively large value of \( L = 2412 \) on all 63 projections. Throughout the simulation results in this paper, the number of outer iterations for our recursive algorithms is set to \( N_{\max} = 3 \) to assure the convergence of the algorithms. Figure 3 suggests a slightly better performance for the MAP decoder for larger values of \( L \). However, surprisingly, our decoding algorithm exhibits a completely opposite behavior, i.e., a better performance is achieved for our subRP A algorithm with smaller values of \( L \). This is then a two-fold gain: a better performance for an encoding scheme that results in a lower complexity for our decoding algorithm. We did extensive sets of experiments which all confirm this empirical observation. However, still, further investigation is needed to precisely characterize the performance-complexity trade-off as a result of the encoding process.

D. Projection Pruning

One direction for reducing the complexity of our decoding algorithms is to prune the number of projections at each layer. Particularly, let us assume that, at each layer and point in the decoding tree diagram, the complexity of decoding each branch (that corresponds to a given projection) is the same. This is not precisely true given that the projected codes at the bottom layer may have different dimensions. Also, we assume that the complexity of the aggregations performed at each layer is the same. Then, pruning the number of projections by a factor \( \beta \in (0,1) \) is roughly equivalent to reducing the complexity by a factor of \( \beta \) at each layer. In other words, if we have a subcode of \( RM(m,r) \), then there are \( r-1 \) layers in the decoding tree and hence, the projection pruning exponentially reduces the decoding complexity by a factor of \( \beta^{r-1} \). This is essential to make the decoding of higher order RM subcodes practical. One can also opt to choose a constant number of projections per layer (i.e., prune the number of projections at upper layers with smaller factors) to avoid high-degree polynomial complexities.

Given that the projected codes at the bottom layer can have different dimensions (in contrast to RM codes), the projection subspaces should be carefully selected to reduce the complexity without having a notable effect on the decoding performance. Our empirical results show that the choice of the sets of projections can significantly affect the decoding performance. To see that, in Figure 4 we consider the generator matrix \( G_{\min,15} \) for encoding a \((64,14)\) RM subcode. In addition to full-projection decoding (i.e., 63 1-D subspaces), we also evaluate the performance of subRP A and soft-subRP A with 15 projections picked according to three different projection pruning schemes. First, we consider a subset of 15 subspaces that results in maximum ranks for the projected generator matrices at the bottom layer. In this setting, denoted by “maxRank” in Figure 4, all the 15 projections result in the same rank of 6. It is observed that this selection of the projections significantly degrades the performance (almost 1 dB gap with full-projection decoding). Our extensive simulation results with other generator matrices and code parameters also confirm the same observation that, although it requires a higher complexity for MAP or soft-MAP decoding of the projected codes at the bottom layer, the “maxRank” selection fails to achieve a good performance compared to other considered pruning schemes.

Next, we consider the other extreme of projection selection, i.e., we select 15 subspaces that result in minimum ranks for the projected codewords (“minRank” scheme in Figure 4). In this case, three of the ranks are equal to 2 and the remaining are equal to 3. Therefore, the decoder in this case can perform the MAP and soft-MAP decodings at the bottom layer almost 8 times faster than in maxRank selection. Surprisingly, despite its lower complexity compared to the maxRank selection, the minRank selection is capable of achieving very close to the performance of full-projection decoding \((\approx 0.1 \text{ dB gap for both subRP A and soft-subRP A})\). Our additional simulation results also confirm the same observation and hence, establish the promising advantages of minRank projection pruning scheme in significantly reducing the decoding complexity while maintaining a negligible gap with the performance of full-projection decoding.

Even though minRank selection scheme is capable of achieving very close to the performance of full-projection decoding, we cannot guarantee that it is the best selection in terms of minimizing the decoding error rate. In practice,
we may want to prune most of projections per layer to allow efficient decoding at higher rates (equivalently, higher order RM subcodes) with a manageable complexity. In such scenarios, we may, inevitably, have a meaningful gap with full-projection decoding, more than what we observed here for minRank selection (i.e., $\approx 0.1$ dB). Therefore, one needs to ensure that the sets of the selected projections are the ones that minimize the decoding error rate, i.e., the gap to the full-projection decoding. In the next subsection, we shed light on how the proposed soft-subRPA algorithm enables training a ML model to search for optimal sets of projections. This then establishes that the combination of our soft-subRPA with our ML model enables efficient decoding (in terms of both decoding error rate and complexity) of RM subcodes. To see the potentials of this scheme, in Figure 4 we included the results of our decoding algorithms with 15 projections obtained via training our ML model. It seems that the trained model also has tendency to pick projections that result in smaller ranks for the projected generator matrices, i.e., 3 rank-2, 6 rank-3, and 6 rank-4 projections are picked by the ML model. Figure 4 demonstrates identical performance to full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding, for both subRPA and soft-subRPA algorithms, which is the best one can hope for with the full-projection decoding.

E. Training an ML Model for Projection Pruning

As explained earlier, the goal is to train an ML model to find the best subset of projections. To do so, we assign a weight metric $w_q$ to each $q$-th projection such that $w_q \in [0, 1]$ and $\sum_{q=1}^{Q} w_q = 1$, where $Q$ is the number of full projections for a given (projected) code in the decoding process. The objective is then to train an ML model to pick a subset of $Q_0$ projections (i.e., prune the number of projections by a factor $\beta = Q_0/Q$) that minimize the training loss. Building upon the success of stochastic gradient descent methods in training complex models, we want to use gradients for this search. In other words, the ML model updates the weight vector $w := (w_q, q \in [Q])$ such that picking the $Q_0$ projections corresponding to the largest weights results in the best performance.

There are two major challenges in training the aforementioned ML model. First, the MAP decoding that needs to be performed at the bottom layer (see (5)) is not differentiable since it involves the $\text{argmax}(\cdot)$ operation which is not a continuous function. Therefore, one cannot apply the gradient-based training methods to our subRPA algorithm. However, the proposed soft-subRPA algorithm overcomes this issue by replacing the non-differentiable MAP decoder at the bottom layer with the differentiable soft-MAP decoder. The second issue is that the combinatorial selection of $Q_0$ largest elements of the vector $w$ is not differentiable. To address this issue, we apply the SOFT (Scalable Optimal transport-based differentiable) top-$k$ operator, proposed very recently in [18], to obtain a smoothed approximation of the top-$k$ operator whose gradients can be efficiently approximated. It is worth mentioning that the SOFT top-$k$ function is a generalization of the soft-max function, which is a soft version of the $\text{argmax}$ function. In other words, the SOFT top-$k$ function can be viewed as a soft version of top-$k$ function.

Next, the training procedure is briefly explained. We use the PyTorch library of Python to first implement our soft-subRPA decoding algorithm in a fully differentiable way for the purpose of gradient-based training. We initialize the weight vector as $w_0 := (1/Q, \cdots, 1/Q)$, i.e., equal weights for all the projections. For each training iteration, we randomly generate a batch of $B$ codewords of the RM subcode, and compute their corresponding LLR vectors given a carefully chosen training SNR. Then we input these LLR vectors to our decoder to obtain the soft decisions at each layer. During the soft-aggregation step, instead of unweighted averaging of (7), we take the weighted averages of the soft decisions at all $Q$ projections as $\hat{I}(z) = \sum_{q=1}^{Q} w_q \text{tanh} \left( \hat{I}_q ((z + B_q) / 2) \right)$, ideally, the top-$k$ operator should return nonzero weights only for the top $Q_0$ elements. However, due to the smoothed SOFT top-$k$ operator, all $Q$ elements of $w$ may get nonzero weights though the weights for the $Q - Q_0$ smaller elements are very small. Therefore, the above weighted average is approximately equal to the weighted average over only the largest $Q_0$ weights (i.e., pruned-projection decoding). Note that we apply the same procedure for all (projected) RM subcodes at each node and layer of the recursive decoding algorithm while we define different weight vectors (and also $Q_0$’s) for each sets of projections corresponding to each (projected) codes. We also consider fixed weight vectors for decoding all $B$ codewords at each iteration.

The ML model then updates all weight vectors at each iteration to iteratively minimize the training loss. To do so, we apply the “Adam” optimization algorithm [19] to minimize

\[ \text{max}(0, x) := \text{max}(0, x) \]
the training loss while using “BCEWithLogitsLoss” \cite{20} as the loss function which efficiently combines a sigmoid layer with the binary cross-entropy (BCE) loss. By computing the loss function between the true labels from the generated codewords and the predicted LLRs from the decoder output, the optimizer then moves one step forward by updating the model, i.e., the weight vectors. Finally, once the model converges after enough number of iterations, we save the weight vectors for the sake of optimal projection pruning. Note that in order to reduce the decoding complexity and the overload of training process, we only train the model for a given, properly chosen, training SNR. In other words, once the training is completed, we fix the subsets of projections according to the largest values of the weight vectors. We then test the performance of our algorithms given the fixed decoder (i.e., the fixed subsets of projections) for all codewords and all SNR points. One can apply the same procedure to train the model for each SNR point, or even actively for each LLR vector, to possibly improve upon the performance of our fixed projection pruning scheme at the expense of increased training overload.

IV. CONCLUSIONS

In this paper, we designed efficient decoding algorithms for decoding subcodes of RM codes. More specifically, we first proposed a general recursive algorithm, called subRPA, for decoding RM subcodes. Then we derived a soft-decision based version of our algorithm, called soft-subRPA, that not only improved upon the performance of the subRPA algorithm but also enabled a differentiable implementation of our decoding algorithm for the purpose of training a machine learning model. Accordingly, we proposed an efficient pruning scheme that finds the best subsets of projections via training a machine learning model. Our simulation results on a (64, 14) RM subcode demonstrate as good as the performance of full-projection decoding for our machine learning-aided decoding algorithms with more than 4 times smaller number of projections. The research in this paper can be extended in many directions such as training machine learning models to design efficient encoders for RM subcodes and also leveraging higher dimension subspaces for projections to, possibly, further reduce the decoding complexity.

APPENDIX A

LLRS OF THE INFORMATION BITS

Consider an AWGN channel model as $y = s + n$, where $s = 1 - 2c$, $c \in C$, and $n$ is the AWGN vector with mean zero and variance $\sigma^2$ elements. Then, the LLR of the $i$-th information bit $u_i$ can be obtained using max-log approximation as

$$I_{\text{inf}}(i) := \ln \left( \frac{\Pr(u_i = 0 | y)}{\Pr(u_i = 1 | y)} \right)$$

where $l := 2y/\sigma^2$ is the LLR vector of the AWGN channel, and $C^0_i$ and $C^1_i$ are subsets of codewords that have the $i$-th information bit $u_i$ equal to zero or one, respectively. To see this, observe that

$$I_{\text{inf}}(i) := \ln \left( \Pr(u_i = 0 | y) \right)$$

(a) $\approx \ln \left( \frac{\sum_{s \in C^0_i} \exp \left( -\frac{1}{2} \frac{|y - s|^2}{\sigma^2} \right)}{\sum_{s \in C^1_i} \exp \left( -\frac{1}{2} \frac{|y - s|^2}{\sigma^2} \right)} \right)$

$$\approx \frac{1}{\sigma^2} \min_{s \in C^0_i} |y - s|^2 - \frac{1}{\sigma^2} \min_{s \in C^1_i} |y - s|^2,$$

where step (a) is by applying the Bayes’ rule, the assumption $\Pr(u_i = 0) = \Pr(u_i = 1)$, the law of total probability, and the distribution of Gaussian noise. Moreover, step (b) is by the max-log approximation. Finally, given that all $s$’s have the same norm, we obtain \cite{9}.

REFERENCES

[1] I. Reed, “A class of multiple-error-correcting codes and the decoding scheme,” Transactions of the IRE Professional Group on Information Theory, vol. 4, no. 4, pp. 38–49, 1960.
[2] D. E. Muller, “Application of Boolean algebra to switching circuit design and to error detection,” Transactions of the IRE professional group on electronic computers, no. 3, pp. 6–12, 1954.
[3] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” IEEE Trans. Inf. Theory, vol. 55, no. 7, pp. 3051–3073, 2009.
[4] S. Kudekar, S. Kumar, M. Mondelli, H. D. Pfister, E. Şaşoğlu, and R. L. Urbanke, “Reed-Muller codes achieve capacity on erasure channels,” IEEE Trans. Inf. Theory, vol. 63, no. 7, pp. 4298–4316, 2017.
[5] E. Abbe, A. Shpilka, and A. Wigderson, “Reed-Muller codes for random erasures and errors,” IEEE Trans. Inf. Theory, vol. 61, no. 10, pp. 5229–5252, 2015.
[6] E. Abbe, A. Shpilka, and M. Ye, “Reed-Muller codes: Theory and algorithms,” arXiv preprint arXiv:2002.03317 [2020].
[7] T. Kaufman, S. Lovett, and E. Porat, “Weight distribution and list-decoding size of Reed-Muller codes,” IEEE Trans. Inf. Theory, vol. 58, no. 5, pp. 2689–2696, 2012.
[8] H. Hassani, S. Kudekar, O. Ordentlich, Y. Polyanskiy, and R. Urbanke, “Almost optimal scaling of Reed-Muller codes on BEC and BSC channels,” in 2018 IEEE Int. Symp. Inf. Theory (ISIT). IEEE, 2018, pp. 311–315.
[9] I. Dumer, “Recursive decoding and its performance for low-rate Reed-Muller codes,” IEEE Trans. Inf. Theory, vol. 50, no. 5, pp. 811–823, 2004.
[10] -----, “Soft-decision decoding of Reed-Muller codes: a simplified algorithm,” IEEE Trans. Inf. Theory, vol. 52, no. 3, pp. 954–963, 2006.
[11] I. Dumer and K. Shabunov, “Soft-decision decoding of Reed-Muller codes: recursive lists,” IEEE Trans. Inf. Theory, vol. 52, no. 3, pp. 1260–1266, 2006.
[12] B. Sarkkou, “Decoding of second order Reed-Muller codes with a large number of errors,” in IEEE Inf. Theory Workshop. IEEE, 2005, pp. 3–pp.
[13] R. Saptharishi, A. Shpilka, and B. L. Volk, “Efficiently decoding Reed-Muller codes from random errors,” IEEE Trans. Inf. Theory, vol. 63, no. 4, pp. 1954–1960, 2017.
[14] E. Santi, C. Hager, and H. D. Pfister, “Decoding Reed-Muller codes using minimum-weight parity checks,” in 2018 IEEE Int. Symp. Inf. Theory (ISIT). IEEE, 2018, pp. 1296–1300.
[15] M. Ye and E. Abbe, “Recursive projection-aggregation decoding of Reed-Muller codes,” IEEE Trans. Inf. Theory, vol. 66, no. 8, pp. 4948–4965, 2020.
[16] D. Fathollahi, N. Farsad, S. A. Hashemi, and M. Mondelli, “Sparse multi-decoder recursive projection aggregation for Reed-Muller codes,” 2020.
[17] F. J. MacWilliams and N. J. A. Sloane, The theory of error correcting codes. Elsevier, 1977, vol. 16.
[18] Y. Xie, H. Dai, M. Chen, B. Dai, T. Zhao, H. Zha, W. Wei, and T. Pfister, “Differentiable top-k with optimal transport,” Advances in Neural Information Processing Systems, vol. 33, 2020.
[19] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980 [2014].
[20] https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html accessed: 2021-01-26.