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, 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 that minimize 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, i.e., the Kronecker powers of a 2×2 matrix, though by different rules for selecting rows. In fact, the construction of polar codes is channel-specific while RM codes have a universal construction. Additionally, RM codes achieve the Shannon capacity of binary erasure channels (BECs) at any constant rate [4], and that of binary symmetric channels (BSCs) at extremal rates [5]. The long-time belief that RM codes achieve the capacity of binary-input memoryless symmetric (BMS) channels, however, still remains an open problem [6]. RM codes are also conjectured to have characteristics similar to those of random codes in terms of 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]–[15], Dumer's recursive list decoding algorithm [10]–[12] is capable of achieving close to maximum likelihood decoding performance for large enough, e.g., exponential in blocklength, list sizes. Recently, Ye and Abbe [16] 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 after a proper aggregation. Very recently, building upon the projection pruning idea in [16], a method for reducing the complexity of the RPA algorithm has also been explored in [17]. Besides lacking an efficient decoder, the structure of RM codes does not allow choosing a flexible rate. In fact, as it will be clarified in Section III, given the code blocklength n, one can only construct RM codes with $1 + \log_2 n$ possible values for the code rate.

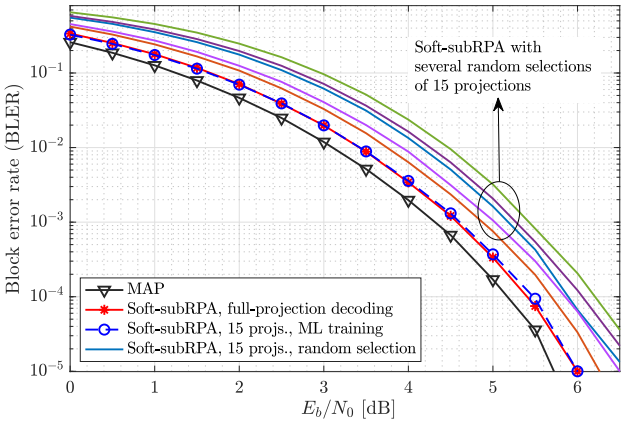

Figure 1. Performance comparison of the MAP decoder with full- and pruned-projection soft-subRPA decoding for a $(64, 14)$ RM subcode.

This research is inspired by the aforementioned 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 them. To this end, we first extend the RPA algorithm to what we call "subRPA" in this paper. 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 *good* sets of projections. We also empirically investigate encoding of RM subcodes.

Figure 1 demonstrates the potentials of our ML-aided soft decoding algorithm, i.e., soft-subRPA with ML-aided projection pruning, in efficiently decoding RM subcodes. In this case study,

an RM subcode with dimension $k = 14$ and blocklength $n = 64$ is considered ($\boldsymbol{G}_{\min,15}$, defined in Section III-C, is used as the generator matrix). Our ML-based pruned-projection decoding, with only 15 projections, is able to achieve an almost identical performance to that of full-projection decoding with 63 projections. This is equivalent to reducing the complexity by a factor of 4, approximately, without sacrificing the performance. Our low-complexity ML-based pruned-projection decoding has then only about 0.25 dB gap with the performance of the optimal maximum a posteriori (MAP) decoding while randomly selecting the subsets of projections does not often provide a competitive performance.

II. Preliminaries

A. RM Codes

Let k and n denote the code dimension and blocklength, respectively. Also, let $m = \log_2 n$. The r-th order RM code of length 2^m, denoted as $\mathcal{RM}(m, r)$, is defined by the following set of vectors as the basis

$$\{\boldsymbol{v}_m(\mathcal{A}) : \mathcal{A} \subseteq [m], |\mathcal{A}| \leqslant r\}, \tag{1}$$

where $[m] := \{1, 2, \ldots, m\}$, $|\mathcal{A}|$ denotes the size of the set $\mathcal{A}$, and $\boldsymbol{v}_m(\mathcal{A})$ is a row vector of length 2^m whose components are indexed by binary vectors $\boldsymbol{z} = (z_1, z_2, \ldots, z_m) \in \{0, 1\}^m$ and are defined as $\boldsymbol{v}_m(\mathcal{A}, \boldsymbol{z}) = \prod_{i\in\mathcal{A}} z_i$. It can then be observed from (1) that $\mathcal{RM}(m, r)$ has a dimension of $k := \sum_{i=0}^{r} \binom{m}{i}$.

According to (1), the (codebook of) $\mathcal{RM}(m, r)$ code is defined as the following set of binary vectors

$$\mathcal{RM}(m, r) := \left\{ \sum_{\mathcal{A}\subseteq[m],|\mathcal{A}|\leqslant r} u(\mathcal{A})\boldsymbol{v}_m(\mathcal{A}) : u(\mathcal{A}) \in \{0, 1\} \right\}. \tag{2}$$

Therefore, considering a polynomial ring $\mathbb{F}_2[Z_1, Z_2, \ldots, Z_m]$ of m variables, the components of $\boldsymbol{v}_m(A)$ are the evaluations of the monomial $\prod_{i\in\mathcal{A}} Z_i$ at points $\boldsymbol{z}$ in the vector space $\mathbb{E} := \mathbb{F}_2^m$. Moreover, each codeword $\boldsymbol{c} = (\boldsymbol{c}(\boldsymbol{z}), \boldsymbol{z} \in \mathbb{E}) \in \mathcal{RM}(m, r)$, indexed by the binary vectors $\boldsymbol{z}$, is defined as the evaluations of an m-variate polynomial with degree at most r at points $\boldsymbol{z} \in \mathbb{E}$.

B. RPA Decoding Algorithm

The RPA algorithm is comprised of the following three building blocks/operations [16].

1) Projection: Considering $\mathbb{B}$ as a s-dimensional subspace of $\mathbb{E}$, with $s \leqslant r$, the quotient space $\mathbb{E}/\mathbb{B}$ contains all the cosets of $\mathbb{B}$ in $\mathbb{E}$. Each coset $\boldsymbol{\tau}$ has the form $\boldsymbol{\tau} = \boldsymbol{z} + \mathbb{B}$ for some $\boldsymbol{z} \in \mathbb{E}$. The RPA algorithm then starts by projecting the log-likelihood ratio (LLR) vector $\boldsymbol{l}$ of the channel output into the subspaces of $\mathbb{E}$. For a one-dimensional (1-D) subspace $\mathbb{B}$, the projected LLR vector can be obtained as $\boldsymbol{l}_{/\mathbb{B}} := (\boldsymbol{l}_{/\mathbb{B}}(\boldsymbol{\tau}), \boldsymbol{\tau} \in \mathbb{E}/\mathbb{B})$, where

$$\boldsymbol{l}_{/\mathbb{B}}(\boldsymbol{\tau}) = \ln\Big(\exp\big(\sum_{\boldsymbol{z}\in\boldsymbol{\tau}} \boldsymbol{l}(\boldsymbol{z})\big) + 1\Big) - \ln\Big(\sum_{\boldsymbol{z}\in\boldsymbol{\tau}} \exp(\boldsymbol{l}(\boldsymbol{z}))\Big). \tag{3}$$

2) Decoding the Projected Outputs: Once the decoder projects the channel output, 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. It then applies the fast Hadamard transform (FHT) [18] to efficiently decode order-1 codes. Afterward, 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 that the algorithm converges.

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. The channel output at a given *point* can be either the actual channel output 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 [16] for one- and two-dimensional subspaces.

III. Efficient Decoding of RM Subcodes

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

Note that the RM encoder does not allow choosing any desired code dimension; it should be of the form $k = \sum_{i=0}^{r} \binom{m}{i}$ for some $r \in \{0, 1, \cdots, m\}$. Suppose that we want to construct a subcode of $\mathcal{RM}(m, r)$ with a dimension k such that $k_l < k < k_u$, where $k_l := \sum_{i=0}^{r-1} \binom{m}{i}$, $r \in [m]$, and $k_u := \sum_{i=0}^{r} \binom{m}{i}$. Given that the construction of RM codes corresponds to picking rows of $\boldsymbol{P}_{n\times n}$ that have the highest Hamming weights, the first k_l rows of the generator matrix $\boldsymbol{G}_{k\times n}$ will be the same as the generator matrix of $\mathcal{RM}(m, r-1)$ that has a Hamming weight of at least 2^{m-r+1}. It then remains to pick extra $k - k_l$ rows from $\boldsymbol{P}_{n\times n}$. These will be picked from the additional $k_u - k_l = \binom{m}{r}$ rows in $\mathcal{RM}(m, r)$ since they all have the next largest Hamming weight of 2^{m-r}. 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 $k - k_l$ 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 study 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 $\boldsymbol{G}_{k\times n}$.

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 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 [16, Eq. (2)]. Additionally, for general BMS channels, the objective is to estimate the projected codewords $\boldsymbol{c}_{/\mathbb{B}}(\boldsymbol{\tau})$'s, $\boldsymbol{\tau} \in \mathbb{E}/\mathbb{B}$, based on the projected LLRs [16]; 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 $k_l < k < k_u$, where $k_l := \sum_{i=0}^{r-1} \binom{m}{i}$, $r \in [m]$, and $k_u := \sum_{i=0}^{r} \binom{m}{i}$. The projection of this code into s-dimensional subspaces of $\mathbb{E}$, $1 \leqslant s \leqslant r-1$, results in subcodes of $\mathcal{RM}(m-s, r-s)$. It is also possible for the projected codes to be $\mathcal{RM}(m-s, r-s)$ or $\mathcal{RM}(m-s, r-1-s)$ codes.*

Proof: Please refer to [19, Proposition 1]. ∎

Hereinafter, for the sake of brevity, we simply state 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 1-D subspaces. However, they can be easily generalized to larger dimension subspaces.

The subRPA algorithm is very similar to the RPA algorithm. In short, it first projects the code $\mathcal{C}$, that is a subcode of $\mathcal{RM}(m,r)$, into 1-D subspaces to get subcodes of $\mathcal{RM}(m-1,r-1)$. 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 [16, 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 and RPA, 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 $\mathcal{RM}(m-r+1,1)$ 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 $\boldsymbol{G}_{k\times n}$. 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 $\boldsymbol{G}_p^{(t)}$ at the bottom layer, $t \in [T]$, where T is the total number of projected codes at that layer (that depends on the number of layers as well as the number of projections per layer). Now, we can pre-compute the codebook $\mathcal{C}_p^{(t)}$ that contains the 2^{R_t} length-$(n/2^{r-1})$ codewords $\boldsymbol{c}_{p,i_t}^{(t)}$, $i_t \in [2^{R_t}]$, of the t-th projected code at the bottom layer. Finally, given the projected LLR vector $\boldsymbol{l}_p^{(t)}$ of length $n/2^{r-1}$ at the bottom layer, we pick the codeword $\boldsymbol{c}_{p,i^*}^{(t)}$ that maximizes the MAP rule for BMS channels [16], i.e.,

$$\hat{\boldsymbol{y}}_t = \boldsymbol{c}_{p,i^*}^{(t)}, \quad \text{s.t.} \quad i^* = \underset{i_t \in [2^{R_t}]}{\operatorname{argmax}} \ \langle \boldsymbol{l}_p^{(t)}, 1-2\boldsymbol{c}_{p,i_t}^{(t)} \rangle, \tag{4}$$

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

B. Soft-SubRPA Algorithm

In this subsection, we derive the soft-decision based version of the subRPA algorithm, referred to as "soft-subRPA", which obtains soft decisions at the bottom layer instead of performing hard MAP decodings; this process is called "soft-MAP" in this paper. Additionally, the decoder applies a different aggregation rule, referred to as "soft-aggregation" in this paper.

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 $\boldsymbol{G}_p$ of a projected code at the bottom layer with codebook $\mathcal{C}_p$. Also, assume a $2^R \times k$ matrix $\boldsymbol{U}$ that lists all 2^R length-k sequences of bits that produce the codebook $\mathcal{C}_p$ (through modulo-2 matrix multiplication $\boldsymbol{U}\boldsymbol{G}_p$). 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 the R information bits given the projected LLR vector $\boldsymbol{l}_p$. This is done, using [19, 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.

Algorithm 1 Soft-MAP Algorithm for AWGN Channels

Input: The LLR vector $\boldsymbol{l}_p$; the generator matrix $\boldsymbol{G}_p$; the codebook $\mathcal{C}_p$; and the matrix $\boldsymbol{U}$ of the information sequences
Output: Soft decisions (i.e., the updated LLR vector) $\hat{\boldsymbol{l}}$

Set k equal to the number of rows in $\boldsymbol{G}_p$
Initialize $\boldsymbol{l}_{\text{inf}}$ as an all-zero vector of length k
$\tilde{\boldsymbol{C}} \leftarrow 1-2\boldsymbol{C}$ ▷ $\boldsymbol{C}$ is the codebook matrix (in binary)
$\tilde{\boldsymbol{l}} \leftarrow \boldsymbol{l}_p \tilde{\boldsymbol{C}}^T$ ▷ matrix mul. of $\boldsymbol{l}_p$ with the transpose of $\tilde{\boldsymbol{C}}$
for $i = 1, 2, \cdots, k$ **do** ▷ obtaining inf. bits LLRs
  if $\boldsymbol{U}(:,i)$ (the i-th column) is not fixed to 0 or 1 **then**
    $\boldsymbol{l}_{\text{inf}}(i) \leftarrow \max_{i' \in \{i' : \boldsymbol{U}(i',i)=0\}} \tilde{\boldsymbol{l}}(i') \ - \max_{i' \in \{i' : \boldsymbol{U}(i',i)=1\}} \tilde{\boldsymbol{l}}(i')$
  end if
end for
Set n' equal to the number of columns in $\boldsymbol{G}_p$
Initialize $\boldsymbol{l}_{\text{enc}}$ as an all-zero vector of length n'
$\boldsymbol{L} \leftarrow \texttt{repeat}(\boldsymbol{l}_{\text{inf}}^T, 1, n')$ ▷ make n' copies of $\boldsymbol{l}_{\text{inf}}^T$
$\boldsymbol{V} \leftarrow \boldsymbol{L} \odot \boldsymbol{G}_p$ ▷ element-wise matrix multiplication
for $j = 1, 2, \cdots, n'$ **do**
  $\boldsymbol{v} \leftarrow$ vector containing nonzero elements of $\boldsymbol{V}(:,j)$
  $\boldsymbol{l}_{\text{enc}}(j) \leftarrow \prod_{j'} \text{sign}(\boldsymbol{v}(j')) \times \min_{j'} |\boldsymbol{v}(j')|$
end for
$\hat{\boldsymbol{l}} \leftarrow \boldsymbol{l}_{\text{enc}}$
return $\hat{\boldsymbol{l}}$

Once we have the LLRs of the information bits, we can combine them according to the columns of $\boldsymbol{G}_p$ to obtain the LLRs of the encoded bits $\boldsymbol{l}_{\text{enc}}$. Note that the codewords in $\mathcal{C}_p$ are obtained by the multiplication of $\boldsymbol{U}\boldsymbol{G}_p$, 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_i's according to the j-th column of $\boldsymbol{G}_p$. Therefore, we can apply the well-known min-sum approximation to calculate the LLR vector of the coded bits as $\boldsymbol{l}_{\text{enc}} := (\boldsymbol{l}_{\text{enc}}(j), j \in [n'])$, where

$$\boldsymbol{l}_{\text{enc}}(j) = \prod_{i \in \Delta_j} \text{sign}(\boldsymbol{l}_{\text{inf}}(i)) \times \min_{i \in \Delta_j} |\boldsymbol{l}_{\text{inf}}(i)|, \tag{5}$$

where Δ_j is the set of indices defining the nonzero elements in the j-th column of $\boldsymbol{G}_p$. This process is summarized in Algorithm 1.

Finally, given the soft decisions at the bottom layer, the decoder needs to properly aggregate them with the (projected) LLRs. In the following, we first define the "soft-aggregation" scheme as an extension of the aggregation method in [16, Algorithm 4]. Please refer to [19] for the details on the derivation of (6).

Definition 1 (Soft-Aggregation)**.** *Let $\boldsymbol{l}$ be the vector of the channel LLRs, with length $n = 2^m$, at a given layer. Suppose that there are Q 1-D subspaces $\mathbb{B}_q$, $q \in [Q]$, to project this LLR vector at the*

next layer (in the case of full-projection decoding, there are $Q = n-1$ 1-D subspaces). Also, let $\hat{\boldsymbol{l}}_q$'s denote the length-$n/2$ vectors of soft decisions of the projected LLRs according to Algorithm 1. The "soft-aggregation" of $\boldsymbol{l}$ and $\hat{\boldsymbol{l}}_q$'s is defined as a length-n vector $\tilde{\boldsymbol{l}} := (\tilde{\boldsymbol{l}}(\boldsymbol{z}), \boldsymbol{z} \in \mathbb{F}_2^m)$ where

$$\tilde{\boldsymbol{l}}(\boldsymbol{z}) = \frac{1}{Q}\sum_{q=1}^{Q} \tanh\left(\hat{\boldsymbol{l}}_q\left([\boldsymbol{z}+\mathbb{B}_q]\right)/2\right)\boldsymbol{l}(\boldsymbol{z}\oplus\boldsymbol{z}_q). \tag{6}$$

where $\boldsymbol{z}_q$ is the nonzero vector of the 1-D subspace $\mathbb{B}_q$, $\oplus$ denotes the coordinate-wise addition in $\mathbb{F}_2$, and $[\boldsymbol{z}+\mathbb{B}_q]$ is the coset containing $\boldsymbol{z}$ for the projection into $\mathbb{B}_q$.

It is worth mentioning that one can also apply the following equation to update the channel LLR as

$$\tilde{\boldsymbol{l}}_{\text{ls}}(\boldsymbol{z}) = \frac{1}{Q}\sum_{q=1}^{Q} \ln\left(\frac{1+\mathrm{e}^{\hat{\boldsymbol{l}}_q([\boldsymbol{z}+\mathbb{B}_q])+\boldsymbol{l}(\boldsymbol{z}\oplus\boldsymbol{z}_q)}}{\mathrm{e}^{\hat{\boldsymbol{l}}_q([\boldsymbol{z}+\mathbb{B}_q])}+\mathrm{e}^{\boldsymbol{l}(\boldsymbol{z}\oplus\boldsymbol{z}_q)}}\right). \tag{7}$$

The rationale behind (7) follows the arguments used to derive (6) in [19] and then deriving the LLR of the sum of two binary random variables given the LLRs of each of them. Therefore, (7) 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 (7), one can reliably apply our proposed soft-aggregation method in Definition 1.

C. Encoding Insights

In this subsection, we provide some insights on how the design of the encoder can affect the decoding complexity as well as the performance[1]. Throughout the paper, we define the signal-to-noise ratio (SNR) as $\text{SNR} := 1/(2\sigma^2)$ and the ratio of the energy-per-bit E_b to the noise as $E_b/N_0 := n/(2k\sigma^2)$, where σ^2 is the noise variance. Furthermore, all the simulation results are obtained from more than 10^5 trials of random codewords. We also iterate the whole decoding process for our recursive algorithms $N_{\max} = 3$ times to ensure their convergence.

Given that the projected codes at the bottom layer can have different dimensions, 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. In other words, let $\mathcal{L} := \sum_{t=1}^{T} 2^{R_t}$ represent a rough evaluation of the (MAP or soft-MAP) decoding complexity at the bottom layer, i.e., the decoding complexity at that layer is roughly a constant times $\mathcal{L}$. Then, among all $\binom{k_u-k_l}{k-k_l}$ possible selections of the generator matrix $\boldsymbol{G}_{k\times n}$, we can choose the ones that achieve a smaller $\mathcal{L}$. In order to investigate the effect of this methodology on the decoding performance, in Figure 2, we consider four different selections of the generator matrix for the $(64, 14)$ RM subcode. In particular, $\boldsymbol{G}_{\max}$ and $\boldsymbol{G}_{\max 2}$ have the largest and the second largest values of $\mathcal{L} = 2568$ and 2532, respectively. Also, $\boldsymbol{G}_{\min}$ has the minimum value of $\mathcal{L} = 1482$. And, $\boldsymbol{G}_{\min,15}$ has the minimum value of $\sum_t 2^{R_t} = 108$ on 15 projections but a relatively large value of $\mathcal{L} = 2412$ on all 63 projections. Figure 2 suggests a slightly better performance for the MAP decoder for larger values of $\mathcal{L}$. However, surprisingly, our decoding algorithm exhibits a completely opposite behavior, i.e., it achieves a better performance for smaller values of $\mathcal{L}$. 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.

[1]We also investigated the efficiency of RM subcodes by comparing their MAP decoding performance with that of time-sharing (TS) between RM codes. We observed that our RM subcodes with rates $14/64$ and $18/64$ achieve more than 1 dB and 0.4 dB gains, respectively, compared to the TS counterparts. Please refer to [19, Figure 2] and the discussions therein for more details.

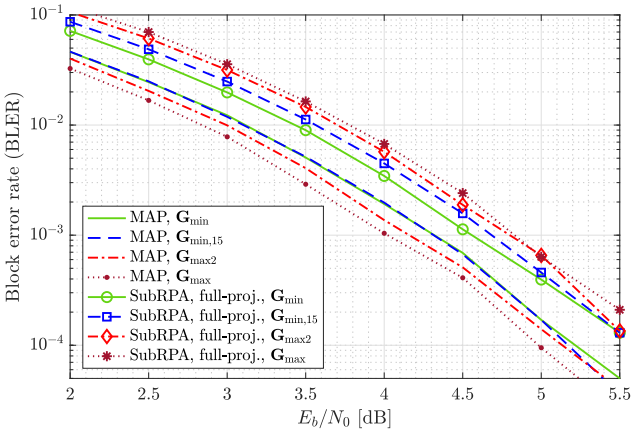

Figure 2. Simulation results for the $(64, 14)$ RM subcodes with MAP and subRPA decoders given four different selections of the generator matrix $\boldsymbol{G}_{k\times n}$.

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 β at each layer. In other words, the projection pruning exponentially reduces the decoding complexity by a factor of β^{r-1} in a subcode of $\mathcal{RM}(m, r)$ that has $r-1$ layers in the decoding tree. 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 to avoid high-degree polynomial complexities.

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 3, we consider the generator matrix $\boldsymbol{G}_{\min,15}$ 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 subRPA and soft-subRPA 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 (referred to as "maxRank" scheme). Figure 3 and our other simulations with different generator matrices and code parameters demonstrate that, although it requires a higher complexity for MAP or soft-MAP decodings at the bottom layer, the maxRank selection fails to achieve a good performance compared to our other considered pruning schemes. Next, we select 15 subspaces that result in minimum ranks for the projected generator matrices ("minRank" scheme in Figure 3). Surprisingly, despite its lower complexity compared to the maxRank selection, the minRank selection is ca-

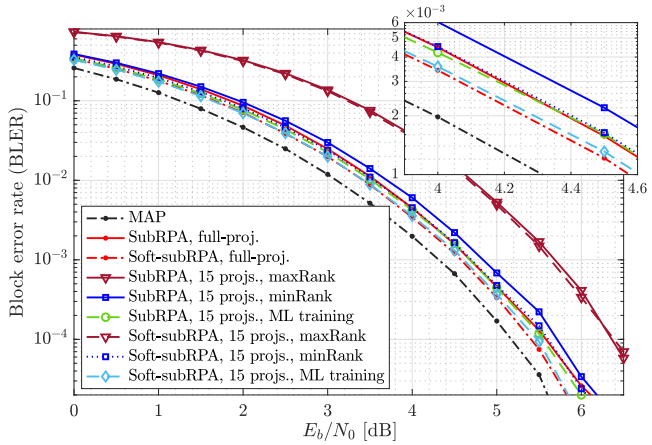

Figure 3. Performance of subRPA and soft-subRPA with full-projection decoding as well as different projection pruning schemes, i.e., picking according to the minimum ranks, maximum ranks, and training a machine learning model. The generator matrix $\boldsymbol{G}_{\min,15}$ is considered for the encoding process.

pable of achieving very close to the performance of full-projection decoding (≈ 0.1 dB gap for both subRPA and soft-subRPA).

In practice, we may prune most of the 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 the minRank selection. 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 an ML model to search for optimal sets of projections. This then establishes that the combination of soft-subRPA with our ML model enables efficient decoding (in terms of both decoding error rate and complexity) of RM subcodes. The results of our decoding algorithms with 15 projections obtained via training our ML model in Figure 3 demonstrate identical performance to full-projection decoding, for both subRPA and soft-subRPA, which is the best one can hope for with the pruned-projection decoding. Additionally, it is observed that the soft-subRPA algorithm can almost 0.1 dB improve upon the performance of subRPA.

E. Training an ML Model for Projection Pruning

The goal is to train an ML model to pick a subset of Q_0 projections out of total number of Q projections (i.e., prune by a factor of $\beta = Q_0/Q$) that minimizes the training loss. To do so, we assign a weight w_q to each q-th projection such that $w_q \in [0, 1]$ and $\sum_{q=1}^{Q} w_q = 1$. Building upon the success of stochastic gradient descent methods in training complex models, we want to use gradients for this search. Indeed, the ML model updates the vector $\boldsymbol{w} := (w_q, q \in [Q])$ such that picking the Q_0 projections with 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 is not differentiable, and thus one cannot apply the gradient-based training methods to our subRPA algorithm. However, the proposed soft-subRPA algorithm overcomes this issue by replacing the MAP decoder with the differentiable soft-MAP decoder[2]. The second issue is that the combinatorial selection of Q_0 largest elements of the vector $\boldsymbol{w}$ is not differentiable. To address this issue, we apply the SOFT (Scalable Optimal transport-based diFferenTiable) top-k operator, proposed very recently in [20], to obtain a smoothed approximation of the top-k operator whose gradients can be efficiently approximated.

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 with 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 (6), we take the weighted averages of the soft decisions at all Q projections as $\tilde{\boldsymbol{l}}(\boldsymbol{z}) = \sum_{q=1}^{Q} w_q \tanh\left(\hat{\boldsymbol{l}}_q\left([\boldsymbol{z} + \mathbb{B}_q]\right)/2\right)\boldsymbol{l}(\boldsymbol{z} \oplus \boldsymbol{z}_q)$. 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 $\boldsymbol{w}$ 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 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. We apply the "Adam" optimization algorithm [21] to minimize the training loss while using "BCEWithLogitsLoss" [22] as the loss function. By computing the loss function between the true labels (from the generated codewords) and the predicted LLRs, 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 the training process, we only train the model for a single SNR point. In fact, once the model is trained, we fix the subsets of projections according to the largest values of the trained weight vectors. We then test the performance of our algorithms given 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 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.

[2]Please refer to [19, Section III-E] for a detailed discussion on why soft-MAP is differentiable but MAP is not.

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, 1954.

[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 and K. Shabunov, "Near-optimum decoding for subcodes of Reed-Muller codes," in *2001 IEEE Int. Symp. Inf. Theory (ISIT)*. IEEE, 2001, p. 329.

[10] 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.

[11] ——, "Soft-decision decoding of Reed-Muller codes: a simplified algorithm," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 954–963, 2006.

[12] 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.

[13] B. Sakkour, "Decoding of second order Reed-Muller codes with a large number of errors," in *IEEE Inf. Theory Workshop, 2005.* IEEE, 2005, pp. 3–pp.

[14] 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.

[15] 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.

[16] 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.

[17] D. Fathollahi, N. Farsad, S. A. Hashemi, and M. Mondelli, "Sparse multi-decoder recursive projection aggregation for Reed-Muller codes," *arXiv preprint arXiv:2011.12882*.

[18] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*. Elsevier, 1977, vol. 16.

[19] M. V. Jamali, X. Liu, A. V. Makkuva, H. Mahdavifar, S. Oh, and P. Viswanath, "Reed-Muller subcodes: Machine learning-aided design of efficient soft recursive decoding," *arXiv preprint arXiv:2102.01671*, 2021.

[20] 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.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] "BCEWithLogitsLoss," https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html, accessed: 2021-01-26.