Machine Learning-Aided Efficient Decoding of Reed–Muller Subcodes

Mohammad Vahid Jamali, *Member, IEEE*, Xiyang Liu, Ashok Vardhan Makkuva, Hessam Mahdavifar, *Member, IEEE*, Sewoong Oh, and Pramod Viswanath

***Abstract*—Reed-Muller (RM) codes achieve the capacity of general binary-input memoryless symmetric channels and are conjectured to have a comparable performance to that of random codes in terms of scaling laws. However, such results are established assuming maximum-likelihood decoders for general code parameters. Also, RM codes only admit limited sets of rates. Efficient decoders such as successive cancellation list (SCL) decoder and recently-introduced recursive projection-aggregation (RPA) decoders are available for RM codes at finite lengths. In this paper, we focus on subcodes of RM codes with flexible rates. We first extend the RPA decoding algorithm to RM subcodes. To lower the complexity of our decoding algorithm, referred to as subRPA, we investigate different approaches to prune the projections. Next, we derive the soft-decision based version of our algorithm, called soft-subRPA, that not only improves upon the performance of subRPA but also enables a differentiable decoding algorithm. Building upon the soft-subRPA algorithm, we then provide a framework for 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 smaller number of projections. We also show that the choice of the projections in decoding RM subcodes matters significantly, and our ML-aided projection pruning scheme is able to find a *good* selection, i.e., with negligible performance degradation compared to the full-projection case, given a reasonable number of projections.**

***Index Terms*—Reed-muller (RM) codes, machine learning, low-complexity decoding, recursive projection-aggregation (RPA) decoding, projection pruning.**

Manuscript received 15 January 2023; revised 30 April 2023 and 12 June 2023; accepted 17 July 2023. Date of publication 25 July 2023; date of current version 10 August 2023. This work was supported in part by the National Science Foundation (NSF) under Grants under Grant CCF-1941633, Grant CCF-2312752, Grant CNS-2002932, and Grant CCF-2312753 and in part by the Office of Naval Research (ONR) under Grant W911NF-18-1-0332. This paper was presented in part at the IEEE International Symposium on Information Theory (ISIT), Melbourne, Victoria, Australia, July 2021 [DOI: 10.1109/ISIT45174.2021.9517885]. *(Corresponding author: Mohammad Vahid Jamali.)*

Mohammad Vahid Jamali and Hessam Mahdavifar are with the Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: mvjamali@umich.edu; hessam@umich.edu).

Xiyang Liu and Sewoong Oh are with the Paul G. Allen School of Computer Science and Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: xiyangl@cs.washington.edu; sewoong@cs.washington.edu).

Ashok Vardhan Makkuva is with the School of Computer and Communication Sciences, EPFL, 1015 Lausanne, Switzerland (e-mail: ashok.makkuva@epfl.ch).

Pramod Viswanath is with the Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ 08540 USA (e-mail: pramodv@princeton.edu).

Digital Object Identifier 10.1109/JSAIT.2023.3298362

I. Introduction

REED-MULLER (RM) codes are among the first families of error-correcting codes, invented almost seven decades ago [2], [3]. They have received significant renewed interest after the breakthrough invention of polar codes [4], given the close connection between the two classes of codes. The generator matrices for both RM and polar codes can be obtained from the same square matrices – the Kronecker powers of a 2×2 matrix – though by different rules for selecting the rows. In fact, such a selection of rows for polar codes is channel-dependent but the RM encoder picks the rows with the largest Hamming weights, resulting in a universal construction. RM codes are also conjectured to have characteristics similar to those of random codes in terms of both weight enumeration [5] and scaling laws [6]. Moreover, Reeves and Pfister have recently shown that RM codes achieve the capacity of general binary-input memoryless symmetric (BMS) channels [7] under the bit maximum-a-posteriori (bit-MAP) decoding. This solves a long-standing open problem in coding theory while leaving the problem of finding efficient decoders for RM codes to provably achieve (or perform close to) such an excellent performance open.

Among the earlier results on decoding RM codes [2], [8], [9], [10], [11], [12], [13], Dumer's recursive list decoding algorithm [8], [9], [10] provides a trade-off between the decoding complexity and the error probability. In other words, it is capable of achieving close to the maximum likelihood decoding performance for large enough, e.g., exponential in blocklength, list sizes. Recently, Ye and Abbe [14] proposed a recursive projection-aggregation (RPA) algorithm for decoding RM codes. The RPA algorithm first projects the received corrupted codeword onto its cosets. It then recursively decodes the projected codes to, finally, construct the decoded codeword by properly aggregating the intermediate decoding results. Building upon the projection pruning idea in [14], a method for reducing the complexity of the RPA algorithm has also been explored in [15]. Moreover, a framework for encoding and decoding RM codes based on the product of smaller RM code components has been explored in [16], with potential applications to low-capacity channels [17]. Furthermore, building upon the computational tree of RM (and polar) codes, a class of neural encoders and decoders has been proposed in [18] via deep learning methods.

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 posses another parameter r, that stands for the *order* of the code, where $0 \leq r \leq m$. Given the code blocklength n, one can then 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 (with flexible rates that can take any code dimension from 1 to n), and our primary goal is to design 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 MAP decoder at the bottom layer, which is feasible and efficient given the low dimension of the projected codes at that layer. It then aggregates the results back to recursively decode the received codeword.

A major focus of this work is on reducing the complexity of our proposed decoding algorithms by pruning many of redundant projections. Through exploring different projection pruning strategies, we empirically show that the choice of projections can significantly impact the decoding performance of RM subcodes. We first propose a method, referred to as the *minRank* projection pruning scheme (incurring the lowest decoding complexity, given a number of projections), that is observed to deliver a very good performance in a variety of scenarios. However, our results show that there are cases where even a random pruning scheme may outperform the minRank selection, especially when the number of projections used for the decoding are significantly smaller than the full number of projections. Motivated by these observations, we leverage the recent advances in channel coding via machine/deep learning [18], [19], [20], [21], [22], [23], [24], [25] to pick the optimal sets of projections via training a machine learning (ML) model. To this end, we first derive the soft-decision based version of the subRPA algorithm, called "soft-subRPA", that not only improves upon the performance of the subRPA algorithm but also provides a differentiable version of our decoding algorithm. Enabled by our differentiable soft-subRPA algorithm, we train an ML model to search for the *good* sets of projections. We find out that carefully training our ML model provides the possibility to find the best sets of projections that achieve very close to the performance of full-projection decoding with much smaller number of projections.

We would like to highlight that our work also adds to the rich literature on soft-decision decoding of algebraic codes, including the celebrated work by Koetter and Vardy on soft-decision decoding of Reed-Solomon codes [26], which is also used for soft-decision decoding of other algebraic codes such as Hermitian codes [27] and elliptic codes [28], as well as the work by Vardy and Be'ery on soft-decision decoding of BoseChaudhuriHocquenghem (BCH) codes [29], among others.

Finally, besides designing efficient decoding algorithms, we also provide some insights on encoding RM subcodes by empirically investigating their performance. Our results show that constructing the code generator matrix with respect to a lower complexity for our algorithms results in a superior performance compared to a higher complexity generator matrix. Also, our empirical results for pruning projections mostly suggest a superior performance for the projection sets incurring a lower decoding complexity. This together with our observation on the encoding part unravels a two-fold gain for our proposed algorithms: a better performance for a lower complexity.

The rest of the paper is organized as follows. In Section II, we provide some preliminaries on RM codes and RPA decoding. In Section III, we present the subRPA and soft-subRPA algorithms for decoding RM subcodes. We empirically investigate encoding of RM subcodes and present several ad-hoc projection pruning schemes in Section IV. Section V is devoted to our ML-aided projection pruning algorithm, and Section VI concludes the paper.

II. Preliminaries

In this section, we briefly review RM codes and the RPA algorithm. The reader is referred to [14] for additional details on the RPA algorithm.

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 by $\mathcal{RM}(m, r)$, is then defined by the following set of vectors as the basis

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

where $[m] \coloneqq \{1, 2, \ldots, m\}$, $|\mathcal{A}|$ 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 $\boldsymbol{z} = (z_1, z_2, \ldots, z_m) \in \{0, 1\}^m$ as

$$\boldsymbol{v}_m(\mathcal{A}, \boldsymbol{z}) = \prod_{i \in \mathcal{A}} z_i, \tag{2}$$

with the convention of $\prod_{i \in \emptyset} z_i \coloneqq 1$. It follows from (1) that $\mathcal{RM}(m, r)$ has the dimension of

$$k = \sum_{i=0}^{r} \binom{m}{i}. \tag{3}$$

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

$$\mathcal{RM}(m, r) \coloneqq \left\{ \sum_{\mathcal{A} \subseteq [m], |\mathcal{A}| \leq r} u(\mathcal{A}) \boldsymbol{v}_m(\mathcal{A}) : u(\mathcal{A}) \in \{0, 1\} \ \forall \mathcal{A} \right\}. \tag{4}$$

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 $\mathbb{E} \coloneqq \mathbb{F}_2^m$. Moreover, each codeword $\boldsymbol{c} = (\boldsymbol{c}(\boldsymbol{z}), \boldsymbol{z} \in \mathbb{E}) \in \mathcal{RM}(m, r)$, that is also 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 main phases.

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) onto the subspaces of $\mathbb{E}$. Considering $\mathbb{B}$ as a s-dimensional subspace of $\mathbb{E}$, with $s \le r$, the quotient space $\mathbb{E}/\mathbb{B}$ contains all the cosets of $\mathbb{B}$ in $\mathbb{E}$. Each coset $\mathcal{T} \in \mathbb{E}/\mathbb{B}$ has the form $\mathcal{T} = \boldsymbol{z} + \mathbb{B}$ for some $\boldsymbol{z} \in \mathbb{E}$. Then, in the case of BSC, the projection of the channel binary output $\boldsymbol{y} = (\boldsymbol{y}(\boldsymbol{z}), \boldsymbol{z} \in \mathbb{E})$ onto the cosets of $\mathbb{B}$ is defined as

$$\boldsymbol{y}_{/\mathbb{B}} \coloneqq \big(\boldsymbol{y}_{/\mathbb{B}}(\mathcal{T}), \mathcal{T} \in \mathbb{E}/\mathbb{B}\big), \ \text{s.t.} \quad \boldsymbol{y}_{/\mathbb{B}}(\mathcal{T}) \coloneqq \bigoplus_{\boldsymbol{z}\in\mathcal{T}} \boldsymbol{y}(\boldsymbol{z}), \quad (5)$$

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 $\boldsymbol{l}$. In the case of a one-dimensional subspace $\mathbb{B}$, the projected LLR vector can be obtained as $\boldsymbol{l}_{/\mathbb{B}} \coloneqq (\boldsymbol{l}_{/\mathbb{B}}(\mathcal{T}), \mathcal{T} \in \mathbb{E}/\mathbb{B})$, where

$$\boldsymbol{l}_{/\mathbb{B}}(\mathcal{T}) = \ln\Big(\exp\big(\sum_{\boldsymbol{z}\in\mathcal{T}} \boldsymbol{l}(\boldsymbol{z})\big) + 1\Big) - \ln\Big(\sum_{\boldsymbol{z}\in\mathcal{T}} \exp(\boldsymbol{l}(\boldsymbol{z}))\Big). \quad (6)$$

In the case of a general s-dimensional subspace $\mathbb{B}$, the quotient space $\mathbb{E}/\mathbb{B}$ contains 2^{m-s} cosets $\mathcal{T}$ each of size 2^s. Then, one can follow a similar approach to the proof of [14, eq. (13)] to prove that $\boldsymbol{l}_{/\mathbb{B}}(\mathcal{T})$, for each coset $\mathcal{T}$, can be obtained recursively as

$$\boldsymbol{l}_{/\mathbb{B}}(\mathcal{T}) = \ln\left(\frac{1 + \exp\big(\boldsymbol{l}_{/\mathbb{B}}\big(\mathcal{T}_{1:2^{s-1}}\big) + \boldsymbol{l}_{/\mathbb{B}}\big(\mathcal{T}_{1+2^{s-1}:2^s}\big)\big)}{\exp\big(\boldsymbol{l}_{/\mathbb{B}}\big(\mathcal{T}_{1:2^{s-1}}\big)\big) + \exp\big(\boldsymbol{l}_{/\mathbb{B}}\big(\mathcal{T}_{1+2^{s-1}:2^s}\big)\big)}\right), \quad (7)$$

where the notation $\mathcal{T}_{i:j}$ is used to denote the subset of $\mathcal{T}$ containing all the elements from index i to j. For the base case of the recursive equation (7) one can use $s = 1$ to obtain (6) as the base case. Alternatively, we can set $s = 0$ as the base case with the convention of $\boldsymbol{l}_{/\mathbb{B}}(\mathcal{T}) \coloneqq \boldsymbol{l}(\boldsymbol{z})$ for a set $\mathcal{T}$ containing a single element $\boldsymbol{z}$. In the latter case, we can derive (6) as a special case of (7) by setting $s = 1$.

2) Decoding the Projected Outputs: Once the decoder projects the channel output ($\boldsymbol{y}$ or $\boldsymbol{l}$), it starts recursively decoding the projected outputs, i.e., it projects them onto new subspaces and continues until the projected outputs correspond to order-1 RM codes. The decoder then applies the fast Hadamard transform (FHT) [30] to efficiently decode order-1 codes. By using the FHT algorithm, one can implement the MAP decoder for the first-order RM codes with the 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 node in the decoding tree), the decoder needs to *aggregate* the output of the channel at that node with the decoding results of the next (underneath) layer to update the channel output. Note that the channel output at a given node can be either the actual channel output ($\boldsymbol{y}$ or $\boldsymbol{l}$) or the projected ones, depending on the depth of that node in the decoding tree of the recursive algorithm. Several aggregation algorithms are presented in [14] for one- and two-dimensional subspaces. We refer the reader to [14] for the details on the aggregation methods.

III. Efficient Decoding of RM Subcodes

A. Problem Setting

An equivalent description of the RM encoder can be obtained through the so-called polarization matrix. Indeed, the generator matrix of an $\mathcal{RM}(m, r)$ code, denoted by $\boldsymbol{G}_{k\times n}$, can be obtained by choosing rows of the following matrix that have a Hamming weight of at least 2^{m-r}:

$$\boldsymbol{P}_{n\times n} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes m}, \quad (8)$$

where $\boldsymbol{F}^{\otimes m}$ is the m-th Kronecker power of a matrix $\boldsymbol{F}$. The resulting generator matrix $\boldsymbol{G}_{k\times n}$ can then be partitioned into sub-matrices as

$$\boldsymbol{G}_{k\times n} = \begin{bmatrix} \boldsymbol{G}_0 \\ \boldsymbol{G}_1 \\ \vdots \\ \boldsymbol{G}_{r-1} \\ \boldsymbol{G}_r \end{bmatrix}, \quad (9)$$

where $\boldsymbol{G}_0$ is a length-n all-one row vector, and $\boldsymbol{G}_1$ is an $m \times n$ matrix that lists all the $n = 2^m$ unique length-m binary vectors $\{0, 1\}^m$ as the columns. Moreover, $\boldsymbol{G}_i$, for $1 \le i \le r$, is an $\binom{m}{i} \times n$ matrix whose each row is obtained by the element-wise product of a distinct selection of i rows from $\boldsymbol{G}_1$ [31]. Accordingly, $\boldsymbol{G}_{k\times n}$ has exactly $\binom{m}{i}$ rows with the Hamming weight $n/2^i$, for $0 \le i \le r$.

As seen, 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 $r \in \{0, 1, \ldots, m\}$. 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 $k_l \coloneqq \sum_{i=0}^{r-1} \binom{m}{i}$ and $k_u \coloneqq \sum_{i=0}^{r} \binom{m}{i}$ for some $r \in [m]$. 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 the lower-order 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 $k - k_l$ rows from $\boldsymbol{P}_{n\times n}$. These will be picked from the additional $k_u - k_l = \binom{m}{r}$ rows in $\boldsymbol{G}_r$ 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 $k - k_l$ rows out of

those $\binom{m}{r}$ rows of weight 2^{m-r} to construct an RM subcode of dimension k as specified above. This important question requires 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 IV-A after describing our decoding algorithms in Sections III-B and III-C with respect to a generic generator matrix $\boldsymbol{G}_{k\times n}$. Our results show that randomly selecting a subset of those rows is not always good. Indeed, some selections are better that the others, and also the set of *good* rows can depend on the underlying decoding algorithm.

B. SubRPA Decoding Algorithm

Before delving 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 (5). Additionally, for general BMS channels, the objective is to estimate the projected codewords $\boldsymbol{c}_{/\mathbb{B}}(\mathcal{T})$'s, $\mathcal{T} \in \mathbb{E}/\mathbb{B}$, based on the channel (projected) LLRs [14]; 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 $k_l := \sum_{i=0}^{r-1}\binom{m}{i}$ and $k_u := \sum_{i=0}^{r}\binom{m}{i}$ for some $r \in [m]$. The projection of this code onto s-dimensional subspaces of $\mathbb{E}$, $1 \le s \le r-1$, 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 Appendix A. ∎

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

As schematically shown in Fig. 1, the subRPA algorithm proceeds in a similar way to the RPA algorithm. More precisely, it first projects the code $\mathcal{C}$, that is a subcode of $\mathcal{RM}(m, r)$, onto 1-D subspaces to get subcodes of $\mathcal{RM}(m-1, r-1)$ 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 [14, 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 the subRPA and RPA algorithms, 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

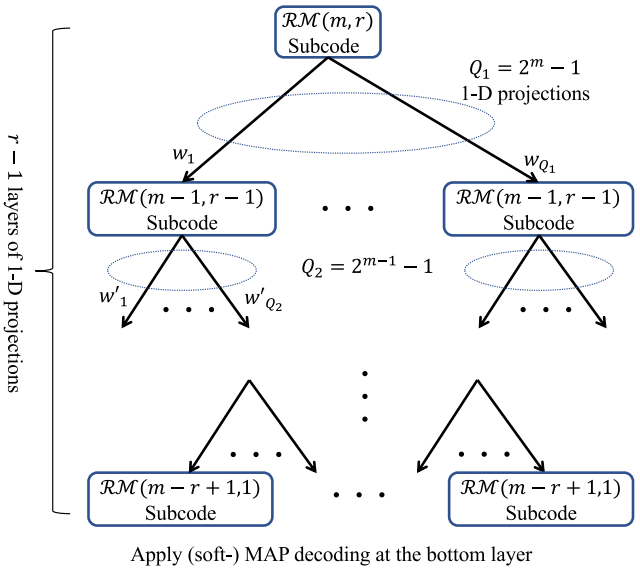

Fig. 1. Schematic diagram of the subRPA and soft-subRPA algorithms.

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 code, 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 the bottom layer (which depends on the number of layers as well as the number of projections per layer). We can then 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. Now, 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 [14], i.e.,

$$\hat{\boldsymbol{y}}_t = \boldsymbol{c}_{p,i^*}^{(t)}, \quad \text{s.t.} \quad i^* = \underset{i_t \in [2^{R_t}]}{\operatorname{argmax}} \left\langle \boldsymbol{l}_p^{(t)}, 1 - 2\boldsymbol{c}_{p,i_t}^{(t)} \right\rangle, \tag{10}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner (dot) product of two vectors. An efficient algorithm for computing $\mathcal{C}_p^{(t)}$ given $\boldsymbol{G}_p^{(t)}$ is presented in Algorithm 2 in Section III-C.

C. Soft-SubRPA Algorithm

In this section, we derive the soft-decision version of the subRPA algorithm, referred to as *soft-subRPA* in this paper. As schematically shown in Fig. 1, the soft-subRPA algorithm obtains soft decisions at the bottom layer instead of performing hard MAP decodings; this process is called *soft-MAP* in

Algorithm 1 Soft-MAP Algorithm for the AWGN Channel

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) $\hat{\boldsymbol{l}}$

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

Algorithm 2 Matrix $\boldsymbol{U}$ and codebook $\mathcal{C}_p$ Finder

Input: The projected generator matrix $\boldsymbol{G}_p$
Output: Matrix $\boldsymbol{U}$ of the information sequences; and codebook $\mathcal{C}_p$ of the projected code

$k \leftarrow$ number of rows in $\boldsymbol{G}_p$
$\mathcal{U}_{\text{ind}} \leftarrow \{\}$ ▷ initialize $\mathcal{U}_{\text{ind}}$ as an empty set
$r \leftarrow 0$
$\boldsymbol{G}_p^{\text{tmp}} \leftarrow [\,]$ ▷ initialize $\boldsymbol{G}_p^{\text{tmp}}$ as an empty matrix
$R \leftarrow \texttt{gfrank}(\boldsymbol{G}_p, 2)$
$i \leftarrow 1$
while $i \leq k$ **and** $r < R$ **do** ▷ iterate over the rows of $\boldsymbol{G}_p$
  Add the i-th row of $\boldsymbol{G}_p$ to $\boldsymbol{G}_p^{\text{tmp}}$
  $i \leftarrow i + 1$
  if $\texttt{gfrank}(\boldsymbol{G}_p^{\text{tmp}}, 2) > r$ **then**
    $r \leftarrow r + 1$
    Add i to $\mathcal{U}_{\text{ind}}$
  end if
end while
$\boldsymbol{U} \leftarrow \boldsymbol{0}_{2^R \times k}$ ▷ initialize $\boldsymbol{U}$ as an all-zero $2^R \times k$ matrix
$\boldsymbol{U}(:, \mathcal{U}_{\text{ind}}) \leftarrow \texttt{de2bi}(0{:}2^R - 1, R)$ ▷ fill out the columns in $\boldsymbol{U}$ indexed by the set $\mathcal{U}_{\text{ind}}$ with the 2^R distinct binary vectors of length R
$\boldsymbol{C} \leftarrow \boldsymbol{U}\boldsymbol{G}_p \bmod 2$ ▷ matrix multiplication over $\mathbb{F}_2$
$\mathcal{C}_p \leftarrow$ rows of $\boldsymbol{C}$ ▷ list all rows of $\boldsymbol{C}$ in $\mathcal{C}_p$
return $\boldsymbol{U}$ and $\mathcal{C}_p$

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 V.

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 the additive white Gaussian noise (AWGN) channel. 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 generate the codebook $\mathcal{C}_p$ (through modulo-2 matrix multiplication $\boldsymbol{U}\boldsymbol{G}_p$).

An efficient algorithm for computing matrix $\boldsymbol{U}$ and codebook $\mathcal{C}_p$ for a given projected generator matrix $\boldsymbol{G}_p$ is presented in Algorithm 2. In Algorithm 2, $\texttt{gfrank}(\boldsymbol{A}, 2)$ is a function that computes the rank of the matrix $\boldsymbol{A}$ over the binary field. Moreover, $\texttt{de2bi}(a : b, m)$ is a function that outputs a $(b - a + 1) \times m$ matrix whose rows are the length-m binary representations of all the integers from a to b. The algorithm first iterates over the rows of $\boldsymbol{G}_p$ to find the index of the (first) R linearly independent rows, i.e., the index of the rows forming a basis for $\boldsymbol{G}_p$. The algorithm stops iterating over the remaining rows as soon as R linearly independent rows are found (i.e., when $r = R$) to avoid unnecessary work. Once the set $\mathcal{U}_{\text{ind}}$ of those indices is found, the $2^R \times k$ matrix $\boldsymbol{U}$ is formed by inserting all distinct binary vectors of length R in the R columns of $\boldsymbol{U}$ indexed by the set $\mathcal{U}_{\text{ind}}$, and freezing the remaining $k - R$ columns to zero. Finally, the codebook $\mathcal{C}_p$ is obtained by the matrix multiplication of $\boldsymbol{U}\boldsymbol{G}_p$ over $\mathbb{F}_2$. The memory required to store the projected generator matrices and codebooks at the bottom layer is quantified in Appendix B.

Given that only R indices of the length-k sequences in $\boldsymbol{U}$ contain the information bits (and the remaining bit positions are frozen to 0), the objective of the first step of the soft-MAP algorithm is to obtain the LLRs of the R information bits using the available projected LLR vector $\boldsymbol{l}_p$. This can be done, using (16) in Appendix C invoking max-log approximation, as described in Algorithm 1. Note that the LLRs of the $k - R$ indices that do not carry information are set to zero.

Once the LLRs of the information bits are calculated, they can be combined according to the columns of $\boldsymbol{G}_p$ to obtain the LLRs of the encoded bits $\boldsymbol{l}_{\text{enc}}$. 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

$$\boldsymbol{l}_{\text{enc}}(j) = \prod_{i \in \Delta_j} \text{sign}(\boldsymbol{l}_{\text{inf}}(i)) \times \min_{i \in \Delta_j} |\boldsymbol{l}_{\text{inf}}(i)|, \tag{11}$$

where Δ_j is the set of indices defining the nonzero elements in the element-wise multiplication of $\boldsymbol{l}_{\text{inf}}$ (to skip the frozen bit positions under the formulation of this paper) with the j-th column of $\boldsymbol{G}_p$. This process is summarized in Algorithm 1 in an efficient way. The decoder may also iterate the whole process several times to ensure 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 [14, Algorithm 4] for the case of soft decisions.

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 $n-1$ 1-D subspaces, hence $Q = n-1$). Also, let $\hat{\boldsymbol{l}}_q$ denote the length-$n/2$ vector 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 $\tilde{\boldsymbol{l}} := (\tilde{\boldsymbol{l}}(z), z \in \mathbb{F}_2^m)$ where

$$\tilde{\boldsymbol{l}}(z) = \frac{1}{Q}\sum_{q=1}^{Q} \tanh\left(\hat{\boldsymbol{l}}_q([z+\mathbb{B}_q])/2\right)\boldsymbol{l}(z \oplus z_q). \tag{12}$$

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 onto $\mathbb{B}_q$.

In order to observe (12), recall that the objective of the aggregation step is to update the length-n channel LLR vector $\boldsymbol{l}$ to $\tilde{\boldsymbol{l}}$ given the soft decisions of the projected codes. $\hat{\boldsymbol{l}}_q([z+\mathbb{B}_q])$ serves as a soft estimate of the binary addition of the coded bits at positions z and $z \oplus z_q$. Hence, by following similar arguments to [14], if that combined bit is 0, then the updated LLR at position z should take the same sign as the channel LLR at position $z \oplus z_q$. Note that this happens with probability $a_0 := 1/[1+\exp(-\hat{\boldsymbol{l}}_q([z+\mathbb{B}_q]))]$. Similarly, with probability $a_1 := 1/[1+\exp(\hat{\boldsymbol{l}}_q([z+\mathbb{B}_q]))]$ the combined bit is 1, and hence the updated LLR at position z and $\boldsymbol{l}(z \oplus z_q)$ should have different signs. Therefore, given a projection subspace $\mathbb{B}_q$, one can update the channel LLR as $a_0 \times \boldsymbol{l}(z \oplus z_q) + a_1 \times -\boldsymbol{l}(z \oplus z_q)$. Taking the average over all Q projections then results in the soft-aggregation rule in (12).

It is worth mentioning that one can also update the channel LLR as

$$\tilde{\boldsymbol{l}}_{\text{ls}}(z) = \frac{1}{Q}\sum_{q=1}^{Q} \ln\left(\frac{1+\mathrm{e}^{\hat{\boldsymbol{l}}_q([z+\mathbb{B}_q])+\boldsymbol{l}(z\oplus z_q)}}{\mathrm{e}^{\hat{\boldsymbol{l}}_q([z+\mathbb{B}_q])}+\mathrm{e}^{\boldsymbol{l}(z\oplus z_q)}}\right). \tag{13}$$

The rationale behind (13) follows by 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, (13) 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 (13), one can reliably apply our proposed soft-aggregation method in Definition 1.

Remark 2: The subRPA and soft-subRPA decoding algorithms reduce to the original RPA decoding algorithm [14] and its soft version, respectively, when applied to an RM code instead of an RM subcode (i.e., when the code dimension k, for a given m, follows Eq. (3)). The only difference is the decoding at the bottom layer, where the FHT decoding can then be directly applied given that all projected codes are order-1 RM codes. Therefore, the proposed ML training approach in Section V can be readily applied to the RM codes as well. However, we will *empirically* establish (see Fig. 10) that the performance of a pruned-projection decoding of an RM code is (almost) the same regardless of the selection of the projections. Therefore, not much (if any) gain can be expected from ML training for projection selection in RPA decoding of RM codes, and simply a random selection of the projections may be sufficient for RPA decoding of RM codes.

Before concluding this section, in the following proposition, we characterize the complexity of our proposed decoding algorithms under different settings

Proposition 2: The decoding complexity of our proposed (soft-) subRPA algorithm in decoding a subcode of an $\mathcal{RM}(m,r)$ code, $r > 1$, is $\mathcal{O}(n^{r-1}\mathcal{C}(m-r+1,1))$, where $\mathcal{C}(m',1)$ stands for the complexity of decoding a subcode of an $\mathcal{RM}(m',1)$ code. Assuming (soft-) MAP at the bottom layer, $\mathcal{C}(m-r+1,1) = \mathcal{O}(n^2/2^{2r-3})$, and the overall decoding complexity simplifies to $\mathcal{O}(n^{r+1})$. The decoding complexity reduces to $\mathcal{O}(n^2)$ for pruned-projection decoding with factor $\beta = \mathcal{O}(1/n)$. The overall complexity further reduces to $\mathcal{O}(n)$ if $2^{R_t} = \mathcal{O}(1)$, $\forall t \in [T]$, in addition to $\beta = \mathcal{O}(1/n)$, where R_t stands for the rank of the t-th projected generator matrix at the bottom layer.

Proof: Please refer to Appendix D. ■

IV. Encoding Insights and Ad-Hoc Projection Pruning

A. Encoding Insights

Although the main objective of this paper is to develop low-complexity schemes for decoding RM subcodes, meanwhile, in this subsection, we provide some insights on how the design of the encoder can affect the decoding complexity as well as the performance. 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 $E_b/N_0 := n/(2k\sigma^2)$, where σ^2 is the noise variance. Additionally, the number of outer iterations for our recursive algorithms is set to $N_{\max} = 3$ to ensure the convergence of the algorithms. In this section, we mainly present the results for relatively short RM subcodes in order to have the ability to obtain the MAP decoding performance for additional insights and comparison. In Section V, we present the results for relatively larger RM subcodes.

First, in order to further highlight the efficiency of RM subcodes, in Fig. 2, we compare the block error rate (BLER) performance of RM subcodes with the performance of time-sharing (TS) between RM codes under 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 (i.e., $\boldsymbol{G}_{\max}$) which will be

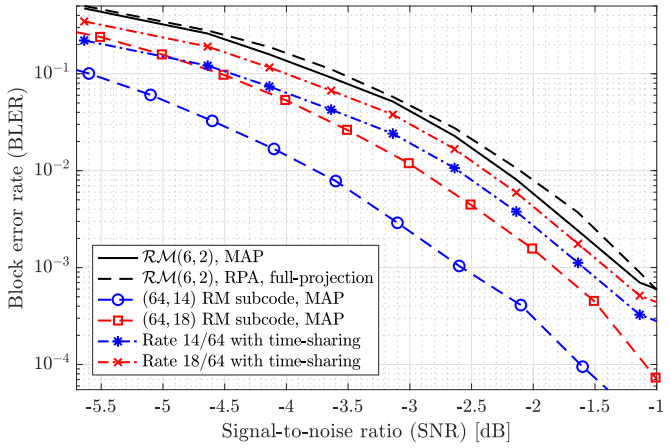

Fig. 2. Simulation results for the BLER of various codes under the MAP decoding. The comparison with the time-sharing scheme between $\mathcal{RM}(6, 1)$ and $\mathcal{RM}(6, 2)$ to achieve the same rates 14/64 and 18/64 is also included.

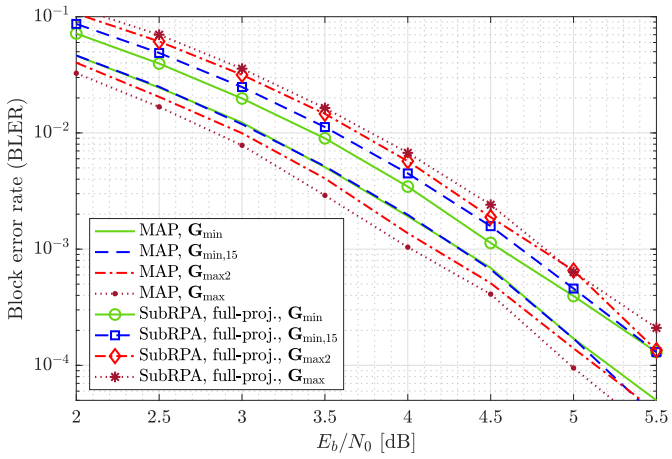

Fig. 3. Simulation results for the (64, 14) RM subcodes under the MAP and subRPA decoding given four different selections of the generator matrix $\boldsymbol{G}_{k\times n}$.

clarified at the end of this subsection. The TS performance is obtained by assuming that the transmitter employs an $\mathcal{RM}(6, 2)$ encoder in α fraction of time and an $\mathcal{RM}(6, 1)$ encoder in the remaining $(1-\alpha)$ fraction. In this experiment, we set $\alpha = 7/15$ and $11/15$ to achieve the same code rates of $14/64$ and $18/64$, respectively, as the RM subcodes. It is observed that the RM subcodes with the rates $14/64$ and $18/64$ achieve more than 1 dB and 0.4 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 10^5 trials of random codewords (except $\mathcal{RM}(6, 2)$ under the MAP decoding that has 10^4 trails).

As discussed earlier, our decoding algorithms perform the MAP or soft-MAP decoding at the bottom layer. Also, the dimension of the projected codes at the bottom layer (i.e., the rank of the projected generator matrices) can be different. This is in contrast to the 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 $L \coloneqq \sum_{t=1}^{T} 2^{R_t}$ 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 $\binom{k_u-k_l}{k-k_l}$ possible selections of the generator matrix $\boldsymbol{G}_{k\times n}$, we can choose the ones that achieve a smaller L. This encoding scheme leads to reduction in the decoding complexity of our algorithms but it can also affect the performance.

In order to investigate the effect of the aforementioned encoding methodology, in Fig. 3, 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 first and second largest values of $L = 2568$ and 2532, respectively, among all possible selections, while $\boldsymbol{G}_{\min}$ having the minimum value of $L = 1482$. Also, $\boldsymbol{G}_{\min,15}$ has the minimum value of $\sum_t 2^{R_t} = 108$ on 15 projections but a relatively large value of $L = 2412$ on all 63 projections. Fig. 3 suggests a slightly better performance under 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 subRPA 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.

B. Ad-Hoc 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 node in the decoding tree, 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. Now, assuming the complexity of the aggregations performed at each layer is the same, 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, if we have a subcode of $\mathcal{RM}(m, r)$, then there are $r-1$ layers in the decoding tree and hence, the projection pruning exponentially reduces the decoding complexity by a factor of β^{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 β's) 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 of RM subcodes. To see this, in Fig. 4, we

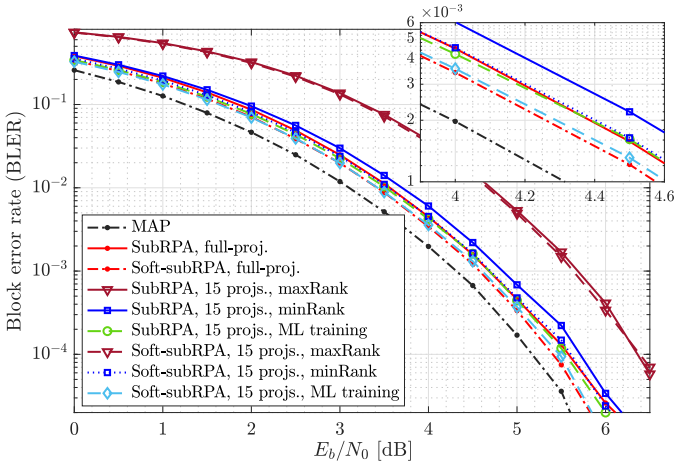

Fig. 4. Performance of subRPA and soft-subRPA under full-projection decoding as well as different projection pruning schemes, i.e., picking according to the minimum ranks, maximum ranks, and training a machine learning model. The generator matrix $\boldsymbol{G}_{\min,15}$ is considered for the encoding process of a (64, 14) RM subcode.

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. In this setting, denoted by "maxRank" in Fig. 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 the MAP or soft-MAP decoding of the projected codes at the bottom layer, the maxRank selection fails to achieve a good performance compared to the other considered projection 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. This proposed method for the selection of projections is referred to as the "minRank" scheme in this paper. 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 9 times faster than the maxRank selection (note that $L = 108$ and 960 for the minRank and maxRank selections, respectively). Surprisingly, despite its lower complexity compared to the maxRank selection, the minRank selection is capable of achieving very close to the performance of the full-projection decoding ($\approx$ 0.1 dB gap in the case of both the subRPA and soft-subRPA decoding). Our additional simulation results – some of which presented in Section V – *mostly* confirm the same observation and suggest a promising performance for the minRank projection pruning scheme or schemes that result in relatively low L's (if not the minimum L).

Even though the minRank selection scheme is capable of achieving very close to the performance of full-projection decoding, one 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. As we will show in Section V, there are scenarios where the performance of the minRank selection significantly diverges from that of the full-projection decoding performance, and it may even perform worse than a random selection of the projections. The failure of the ad-hoc projection pruning schemes in guaranteeing a good performance is the major motivation behind our ML-aided projection pruning scheme presented in the next section.

In the next section, we shed light on how the proposed soft-subRPA algorithm enables training an ML model to search for the optimal set of projections. This will then establish the fact that the combination of our soft-subRPA decoding algorithm with our ML-aided projection pruning framework enables efficient decoding (in terms of both decoding error rate and complexity) of RM subcodes. To see the potentials of this scheme, in Fig. 4 the results of our decoding algorithms with 15 projections picked by training our ML model are also included. It is observed that the trained model also has the 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 (resulting in $L = 156$). Fig. 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 pruned-projection decoding. Additionally, it is observed that the soft-subRPA algorithm can improve upon the performance of the subRPA algorithm by almost 0.1 dB.

V. ML-Aided Projection Pruning

As mentioned earlier, the goal is to train an ML model to find the best subset of projections. To do so, as schematically shown in Fig. 1, 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 $\boldsymbol{w} := (w_q, q \in [Q])$ 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 (10)) is not differentiable since it involves the argmax$(\cdot)$ operation which

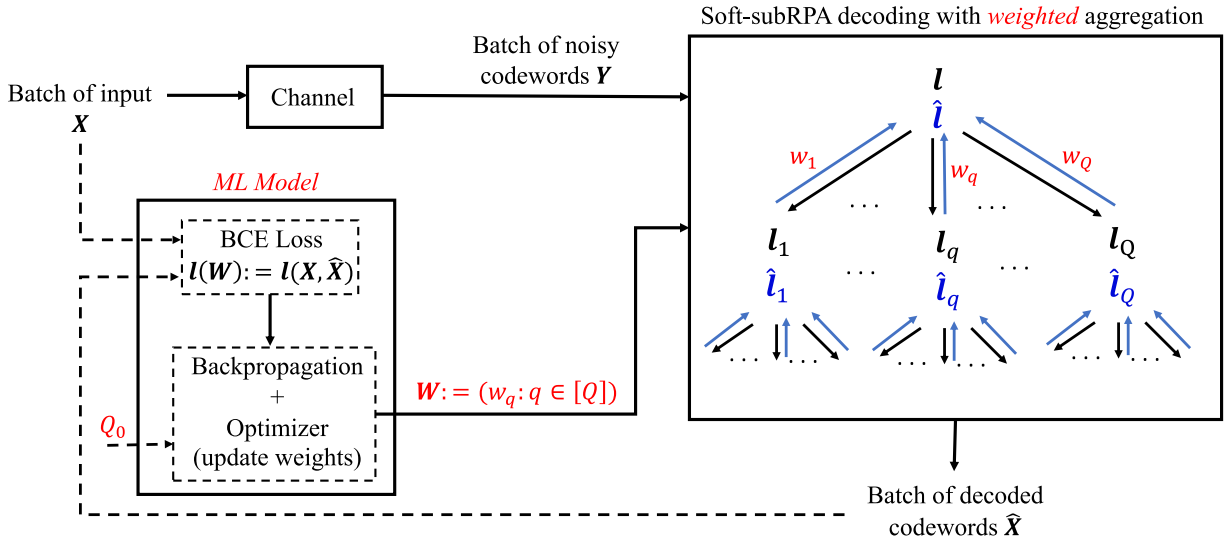

Fig. 5. The training procedure of the proposed ML-aided projection pruning scheme for decoding RM subcodes.

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.[1] 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 [32], 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 argmax function. In other words, the SOFT top-k function can be viewed as a soft version of the top-k function.

The training procedure is schematically shown in Fig. 5, and is briefly explained next. We use the PyTorch library of Python to first implement our soft-subRPA decoding algorithm in a fully differentiable way for the purpose of the gradient-based training.We initialize the weight vector as $\boldsymbol{w}_0 := (1/Q, \ldots, 1/Q)$, 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 the soft-subRPA decoder to obtain the soft decisions at each layer. During the soft-aggregation step, instead of unweighted averaging of (12), the weighted averages of the soft decisions at all Q projections are computed as

$$\tilde{\boldsymbol{l}}(z) = \sum_{q=1}^{Q} w_q \tanh\left(\hat{\boldsymbol{l}}_q\left([z + \mathbb{B}_q]\right)/2\right)\boldsymbol{l}(z \oplus z_q). \tag{14}$$

[1]Note that the soft-MAP algorithm involves $\max(\cdot)$ function which, unlike $\operatorname{argmax}(\cdot)$, is a continuous function. Also, the derivative of the function $\max(0, x)$ is defined everywhere except in $x = 0$ which is a rare event to happen. Accordingly, advanced training tools, such as PyTorch library (that is used in this research), easily handle and treat $\max(\cdot)$ as a differentiable function. For example, the rectified linear unit function $\mathrm{ReLU}(x) := \max(0, x)$ is a widely used activation function in neural networks.

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 major accumulation of weights will be on the largest Q_0 elements once the training is completed. Therefore, the above weighted average is approximately equal to the weighted average over the largest Q_0 weights (i.e., (14) represents a proper approximation of the aggregation in the case of the 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 possibly different 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.

Once the soft decoding of the codewords are obtained, the ML model updates all weight vectors at each iteration to iteratively minimize the training loss. To do so, we apply the "Adam" optimization algorithm [33] to minimize the training loss while using "BCEWithLogitsLoss" [34] as the loss function, which efficiently combines a sigmoid layer with the binary cross-entropy (BCE) loss. By computing the loss 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 to perform 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 decoder by picking only 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 across 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

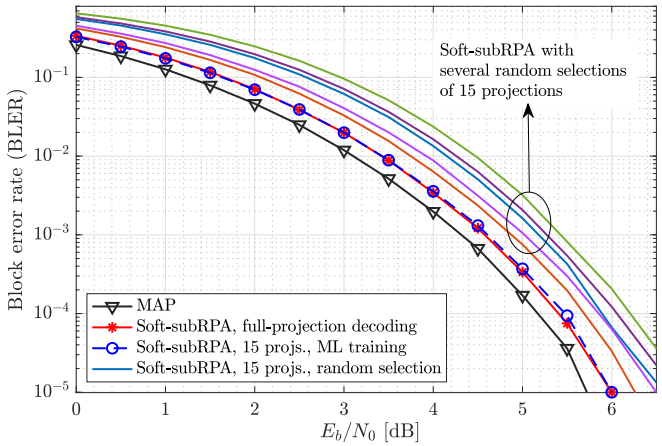

Fig. 6. Performance comparison of the MAP decoder with full- and pruned-projection soft-subRPA decoding for a (64, 14) RM subcode encoded using the generator matrix $\boldsymbol{G}_{\min,15}$. The performance of the ML-aided projection pruning is also compared to several random selections of projections.

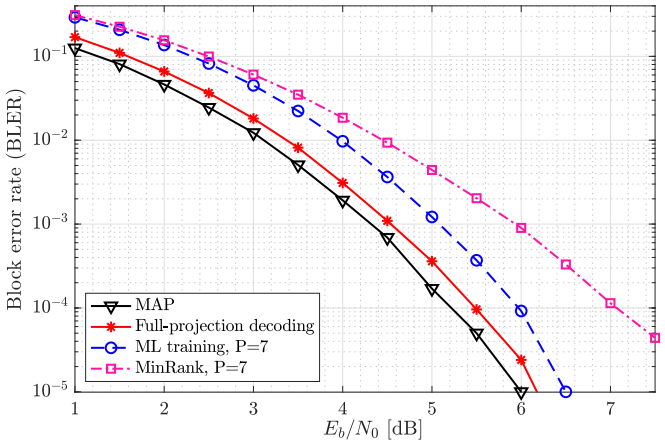

Fig. 7. Performance comparison of the MAP decoder with the soft-subRPA decoding for a (64, 14) RM subcode encoded using the generator matrix $\boldsymbol{G}_{\min}$. Full-projection decoding and pruning with $P = 7$ projections are considered.

improve upon the performance of our *fixed* projection pruning scheme at the expense of increased model complexity and training overload.

The training SNR, which will be used to generate noisy codewords as training data, is an important hyper-parameter that needs to be carefully chosen to ensure a good performance. In the context of training models for channel coding, it is conventional to consider a smaller training SNR for the decoder training schedule compared to the encoder training schedule, as the former is often a more challenging task than the latter. It is also possible to consider a range of training SNR to further help the single trained model to generalize and perform well across a wide range of SNR during the inference phase (see, e.g., [35] for a thorough empirical investigation on how the training SNR affects the training performance of channel encoders and decoders). In this paper, we use a single SNR point (not a range) for training the model to prune the decoding projections. We use the result of the full-projection pruning as a benchmark to select the training SNR point (by considering the pruning effect). Specifically, if the full-projection pruning requires γ dB to achieve the BLER of 10^{-3}, we pick the training SNR as $\gamma + \epsilon$ dB, for some positive offset ϵ that needs to be adjusted according to the pruning factor (i.e., ϵ is larger if a larger fraction of projections are pruned). Note that this heuristic approach is to pick a starting training SNR, and the final training SNR may need to be adjusted by further hyper-parameter tuning.

Fig. 6 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 experiment, $\boldsymbol{G}_{\min,15}$ is used to encode a (64, 14) RM subcode.[2] It is observed that our ML-based projection pruning scheme, with only 15 projections, is able to achieve an almost identical performance to that of the full-projection soft-subRPA decoding with 63 projections. This is equivalent to reducing the complexity by a factor of more than 4 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 MAP decoding. For comparison, the performance of the pruned-projection decoding under several random selections of 15 projections is also provided. As seen, the choice of the projections can significantly impact the decoding performance of RM subcodes, and randomly selecting the subsets of projections cannot guarantee a competitive performance.

Fig. 7 presents the performance of a (64, 14) RM subcode encoded using the generator matrix $\boldsymbol{G}_{\min}$. Pruned-projection soft-subRPA decoding with very small number of projections, i.e., $P = 7$, is considered. The ML-aided projection-pruned decoding, with 9 times smaller number of projections, is observed to have less than 0.4 dB gap with the full-projection decoding. However, the minRank selection significantly degrades the performance, resulting in more than 1 dB gap with the ML-aided pruning scheme at the BLER of 10^{-4}. To train the ML model in Fig. 7, Q_0 was set to 5 during the training phase but $P = 7$ projections corresponding to the largest 7 weights were selected for the testing. The rationale behind this selection was that nearly 20% of the weights were distributed outside the largest 5 weights (due to the SOFT top-k function), as the sorted weight vector after training was $\boldsymbol{w}_{\text{sorted}} = [0.2012,\ 0.1781,\ 0.1519, 0.1444, 0.1279, 0.1277, 0.0689,\ 0.0000, \ldots, 0.0000]$. Out of 63 projected generator matrices of $\boldsymbol{G}_{\min}$, there are 1 with rank 1, 2 with rank 2, 28 with rank 4, and 32 with rank 5. Therefore, the projections picked by the minRank selection scheme result in the set of ranks $\{1, 2, 2, 4, 4, 4, 4\}$. The ML-based selection scheme, however, is observed to pick projections that result in the set $\{2, 4, 4, 4, 4, 4, 4\}$ of ranks, implying that some useful information may be lost if the decoder just picks the projections corresponding to minimum ranks (and thus some higher-rank projections

[2]We should emphasize that the proposed decoding algorithms and the ML-aided projection pruning scheme are presented in general forms and are not restricted to low rates and lengths. While decoding a higher-order RM subcode requires a higher complexity, the ML-aided pruning scheme reduces the complexity by a factor of β^{r-1} ensuring the best decoding performance given a pruning factor. In our numerical experiments, we focus on subcodes of order-2 RM codes that correspond to relatively small code dimensions (i.e., low rates). This should not be interpreted as a limitation of our schemes.

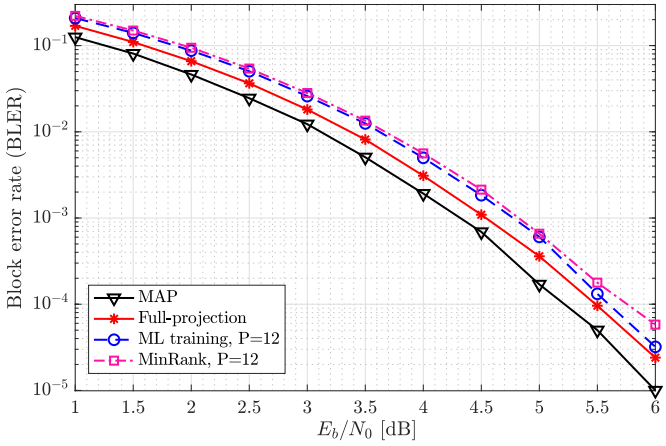

Fig. 8. Performance of the full- and pruned-projection ($P = 12$ projections) soft-subRPA decoding of a (64, 14) RM subcode encoded using $\boldsymbol{G}_{\min}$.

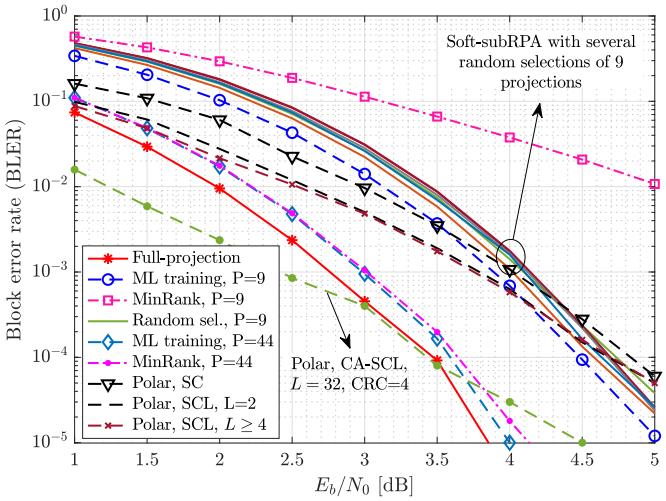

Fig. 9. Performance of the full- and pruned-projection soft-subRPA decoding of a (256, 30) RM subcode generated through the $\boldsymbol{G}_{\min}$ encoding. $P = 9$ and 44 projections are considered for the pruned-projection decoding under the minRank, ML-aided, and random pruning schemes. The plots for the performance of the polar (256, 30) code under successive cancellation (SC) decoding, SC-list (SCL) decoding, and cyclic redundancy check (CRC) aided SCL (CA-SCL) are also included.

are needed) when a significant fraction of projections are pruned.[3]

Fig. 8 shows the performance of a (64, 14) RM subcode, encoded using $\boldsymbol{G}_{\min}$, under the MAP and soft-subRPA decoding. The ML training was performed under $Q_0 = 7$ projections. However, since there were 12 projections with much larger weights, $P = 12$ projections are considered for the testing plots of the pruned-projection decodings in Fig. 8. It is observed that both the minRank and ML-aided pruning schemes achieve very close to the performance of the full-projection decoding, with the ML-aided scheme slightly improving upon the minRank selection at higher SNRs (note that 5×10^5 codewords were used to simulate the performance at each SNR point). In terms of the rank statistics, it is observed that both selection schemes pick the projections that result in the minimum ranks, i.e., 1 rank-1, 2 rank-2, and 9 rank-4 projections are picked by both schemes. However, the set of the selected projections are still different, as the two schemes only have 6 projections in common, out of the total 12 projections. In this case, we can think of the ML model *breaking ties* among the projections that result in the same rank.

Note that the parameter Q_0 is in general a hyper-parameter that needs to be tuned during the training. However, our experiments show that it is not very sensitive, i.e., a model trained for a given Q_0 may work well for different values of P (i.e., the number of projections during testing/inference). In an ideal case, to use a fixed number P of projections for pruned-projection decoding, one can set the parameter $Q_0 = P$ for training. However, this choice may not be the best option. First, due to the SOFT top-k operator, we may not observe a sharp drop of trained weights after exactly Q_0 largest weights. Second, it is possible that some projections are equally good/bad and it is hard for the ML model to perfectly distinguish among them, so the ML model may end up assigning similar weights to such projections. Therefore, to use a fixed P, one can train ML models for some larger/smaller values of Q_0 than P, in addition to $Q_0 = P$. However, our various training experiments (not presented here) suggest that this hyper-parameter tuning does not much affect the performance of the trained model. In the following figure, we use a single model trained for $Q_0 = 20$ for the selection of both $P = 9$ and $P = 44$ projections in an RM subcode of parameters (256, 30).

Fig. 9 presents the results for a medium-length RM subcode of parameters $n = 256$ and $k = 30$ constructed according to the $\boldsymbol{G}_{\min}$ encoding. To train the ML-aided projection-pruning model, Q_0 was set to 20. However, two different values of $P = 9$ and 44 are used as the number of projections for testing the performance. These selections for P were made by taking into account the profile of the weights after training (picking a P if there is a sharp drop in the value of the next largest weight), and to study two extreme scenarios: 1) a relatively small number of projections such that there is a significant gap to the full-projection decoding; and 2) a relatively large P where the performance of the ML-aided pruned-projection decoding is close to that of the full-projection decoding.

When $P = 9$, where the projections are heavily pruned by a factor of more than 28, the minRank training is observed to significantly diverge from the full-projection decoding performance (e.g., nearly 3 and 4 dB gaps at the BLERs of 10^{-2} and 10^{-4}, respectively). However, training the ML model is shown to enable achieving a significantly better performance. Moreover, the performance of several random selections of the projections are also tested, where, similar to Fig. 6, it is observed that the random projection pruning scheme fails to guarantee achieving the best performance for a given value of P. On the other hand, when $P = 44$ projections are used, both the minRank and ML-aided projection pruning

[3]We should emphasize that this does not mean that the ML-based selection scheme favors higher rank projections. Indeed, our extensive experiments suggest that the ML-based selection mostly favors smaller-rank projections. Specifically, it either results in the same set of ranks as the minRank selection or only substitutes some very low-rank projections with (slightly) higher-rank projections.

schemes are observed to achieve very close to the performance of the full-projection decoding, with the ML-aided scheme slightly outperforming the minRank scheme at the higher SNRs.

Fig. 9 also compares the performance of the RM subcode with that of the polar $(256, 30)$ code under successive cancellation (SC) decoding and SC-list (SCL) decoding. To construct the polar code, the Tal-Vardy code construction method is used to pick the k bit-channels with the smallest BERs [36]. The performance of the cyclic redundancy check (CRC) aided SCL (CA-SCL) decoding of the polar code is also included. We note, however, that the comparison to the CA-SCL may not be fair as one can also do RM-CRC and consider RPA-type decoding algorithms together with Chase list decoding (see, e.g., [14]). Indeed, the comparison of plain codes with plain decoders is more meaningful, and polar with CRC is essentially a concatenated design. The following are the main conclusions drawn from this figure.

- First, the polar code under SC decoding fails to provide a comparable performance to that of the RM subcode, even under $P = 7$ projections.
- The performance of the polar code under SCL quickly saturates with respect to the list size L such that only a very minimal improvement is observed with increasing L, i.e., some gains from $L = 1$ to $L = 2$, very little gain from $L = 2$ to $L = 4$, and no gain from $L = 4$ to larger L's. This is while the RM subcode is able to achieve a much better performance by increasing P from 9 to 44.
- The RM subcode under $P = 44$ is able to achieve a significantly better performance than the polar code under SCL decoding with any list size. Even with $P = 9$, the RM subcode beats the polar code under SCL for BLERs smaller than $\approx 7 \times 10^{-4}$.

It is worth noting that, as seen in Fig. 3, the performance of an RM subcode, for a given k and n, highly depends on the selection of the rows, i.e., the encoder design. Therefore, the objective of the paper is not to have a better performance than other classes of codes (which necessitates the best design of the RM subcode encoder) but to deliver the best performance given an RM subcode encoder (that, as shown above, has the potential to beat other classes of codes). We shall emphasize that the low latency of our decoding algorithms is another major advantage compared to polar codes as all decoding branches in the decoding tree (see Fig. 1) can be executed in parallel.

In Fig. 10, the (soft-) subRPA decoding algorithm is applied to an $\mathcal{RM}(6, 2)$ code (that has $k = 22$ and $n = 64$). As discussed in Remark 2, in this case, our decoding algorithm reduces to the original RPA decoding of RM codes [14]. By evaluating the performance of many different random selections of $P = 12$ projections, it is observed that the performance of a pruned-projection decoding of an RM code, for a given P, is (almost) the same regardless of the selection of the projections. This empirical observation then suggests that not much (if any) gain can be expected from ML training for projection selection in RPA decoding of RM codes. As such, our ML-based projection selection as well as the minRank

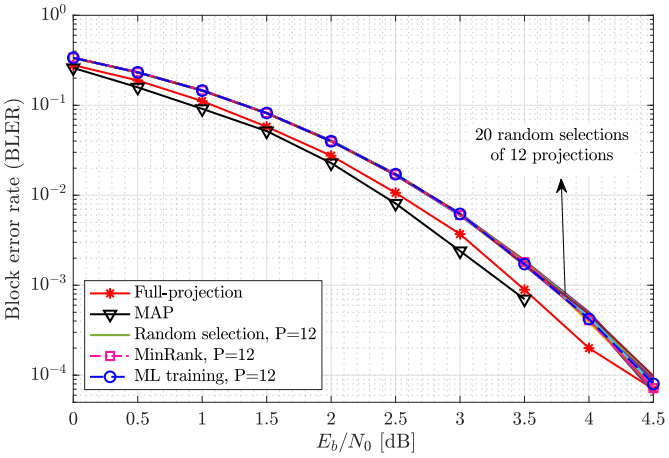

Fig. 10. Performance of the full- and pruned-projection soft-subRPA (that reduces to the soft-version of RPA) decoding of an $\mathcal{RM}(6, 2)$ code. $P = 12$ projections are considered for the pruned-projection decoding under the minRank, ML-aided, and random pruning schemes.

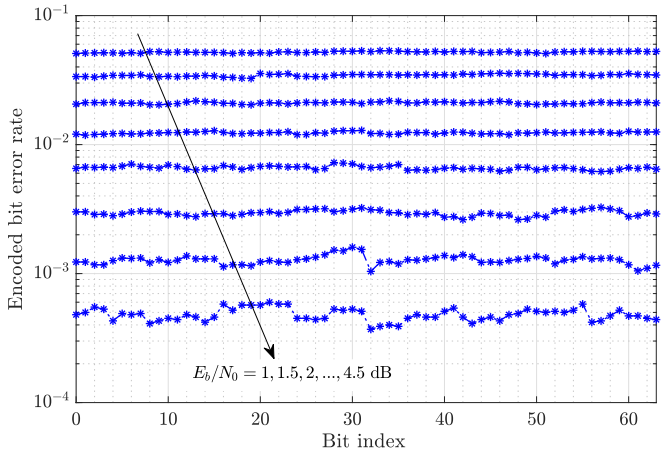

Fig. 11. Encoded bits error rate profile for the $(64, 14)$ RM subcode under ML-aided projection selection.

scheme achieved the same performance as random selection of projections. This further suggests that the selection of projections is strongly tied to the rank profile/properties of the so-called projected generator matrices. We believe the theoretical study of this behavior, on both encoding and decoding of RM subcodes, is an interesting direction for future research.

Finally, Fig. 11 shows the error probability profile of encoded bits for the sample $(64, 14)$ RM subcode with $P = 12$ ML-aided projections that corresponds to the setting in Fig. 8. The E_b/N_0 is changed from 1 dB to 4.5 dB with the step size of 0.5 dB. For each E_b/N_0 point, 10^5 random codewords are examined and the mismatch of the decoder output with the encoded codeword is evaluated. It is observed that under all evaluated E_b/N_0's, all encoded bits experience a relatively uniform/equal error probability.

VI. Conclusion

In this paper, we designed efficient decoding algorithms for decoding subcodes of RM codes. More specifically, we first

proposed a general recursive algorithm, namely the subRPA algorithm, for decoding RM subcodes. Then we derived a soft-decision based version of our algorithm, called the soft-subRPA algorithm, that not only improved upon the performance of the subRPA algorithm but also enabled a differentiable implementation of the decoding algorithm for the purpose of training a machine learning (ML) model. Accordingly, we proposed an efficient pruning scheme that finds the best subsets of projections via training an ML model.

Our simulation results on $(64, 14)$ and $(256, 30)$ RM subcodes demonstrate achieving very close the performance of the full-projection decoding using our ML-aided pruned-projection decoding algorithm with more than 4 times smaller number of projections. Our decoding algorithm also inherits the low-latency and parallelized implementation of the RPA algorithm; when the training is completed, the set of projections are fixed, and all branches in the decoding tree can be executed in parallel. We also provided some insights on encoding RM subcodes and studied several ad-hoc projection pruning schemes. Our extensive simulations showed that the random selection of projections cannot guarantee a competitive performance to that of the ML-aided pruning scheme, while the proposed minRank pruning scheme being *often* a reasonable structured scheme, especially when the projections are not heavily pruned. On the other hand, when a significant fraction of projections are pruned, the minRank scheme was observed to significantly degrade the performance compared to the ML-aided pruning scheme.

The research in this paper can be extended in several directions such as training ML 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
Proof of Proposition 1

The projection of $\mathcal{RM}(m, r)$ onto a s-dimensional subspace, $1 \le s \le r$ is an $\mathcal{RM}(m-s, r-s)$ code [14]. The code $\mathcal{C}$, that is a subcode of $\mathcal{RM}(m, r)$, 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 $\mathcal{RM}(m, r-1)$. We note that the projection of $\mathcal{RM}(m, r-1)$ onto a s-dimensional subspace, $1 \le s \le r-1$, 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 $\mathcal{RM}(m-s, r-1-s)$ 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 $k - k_l$ rows do not contribute in the rank of the merged matrix according to a given subspace, the projected code onto 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 $\mathcal{RM}(m-s, r-s)$. ∎

Appendix B
Memory Requirements to Store Projected Matrices in (Soft-) MAP Algorithm

As discussed in Sections III-B and III-C, one can precompute and store the codebook of each projected code at the bottom layer to facilitate the (soft-) MAP decoding at that layer. In this appendix, we quantify the memory requirement for storing such matrices at the bottom layer, and discuss alternative approaches in applications with limited memory availability.

Recall that for a subcode of $\mathcal{RM}(m, r)$, with $r > 1$, the decoding involves $r-1$ layers of 1-D projections, resulting in $T = \prod_{i=1}^{r-1}(\frac{n}{2^{i-1}} - 1) = \mathcal{O}(n^{r-1})$ projections for full-projection decoding. This number reduces to $T = \mathcal{O}(\beta^{r-1} n^{r-1})$ for a pruned-projection decoding with the pruning factor $\beta < 1$. After $r-1$ layers of 1-D projections, we arrive at subcodes of $\mathcal{RM}(m-r+1, 1)$ whose dimension is $R_t \le m-r+2$, $\forall t \in [T]$. Therefore, the so-called projected codebooks $\mathcal{C}_p^{(t)}$ will contain 2^{R_t} (i.e., at most $2^{m-r+2} = n/2^{r-2}$) length-$(n/2^{r-1})$ codewords, that can be stored in so-called projected codebook matrices $\boldsymbol{C}_p^{(t)}$ of size at most $(n/2^{r-2}) \times (n/2^{r-1})$. Therefore, $\mathcal{O}(\beta^{r-1} n^{r+1}/2^{2r-3})$ bits are required to store all T projected codebooks. For example, for subcodes of $\mathcal{RM}(6, 2)$ and $\mathcal{RM}(8, 2)$ with $\beta = 7/63$ and $9/255$ (that correspond to Figs. 7 and 9, respectively), at most 14,563 and 296,068 bits (i.e., nearly 1.82 kB and 37 kB) respectively, are needed to store all codebooks at the bottom layer.

Similarly, $\mathcal{O}(k\beta^{r-1} n^r/2^{r-1})$ bits are needed to store all T projected generator matrices $\boldsymbol{G}_p^{(t)}$ of dimension $k \times 2^{m-r+1}$. Finally, since each matrix $\boldsymbol{U}_p^{(t)}$ is of size $2^{R_t} \times k$, $\mathcal{O}(k\beta^{r-1} n^r/2^{r-2})$ bits are also needed to store all T matrices $\boldsymbol{U}_p^{(t)}$. Therefore, the memory M_{tot} (in terms of the number of bits) required to store all matrices $\boldsymbol{C}_p^{(t)}$, $\boldsymbol{U}_p^{(t)}$, and $\boldsymbol{G}_p^{(t)}$, $\forall t \in [T]$, at the bottom layer can be characterized as

$$\begin{aligned} M_{\text{tot}} &= \mathcal{O}\Big(\beta^{r-1} n^{r+1}/2^{2r-3}\Big) + \mathcal{O}\Big(k\beta^{r-1} n^r/2^{r-1}\Big) \\ &\quad + \mathcal{O}\Big(k\beta^{r-1} n^r/2^{r-2}\Big) \\ &= \mathcal{O}\Big(\beta^{r-1} n^r \Big[3k + n/2^{r-2}\Big]/2^{r-1}\Big) \\ &= \mathcal{O}\Big((n\beta/2)^r \Big[k + n/2^{r-2}\Big]\Big). \end{aligned} \tag{15}$$

Note that the pruning factor β can be essentially $\mathcal{O}(1/n)$ so that the number of projections in each layer, i.e., $\mathcal{O}(\beta n)$, becomes a constant. Then $(\beta n)^r = \mathcal{O}(1)$ (though with a large constant) and the overall memory requirement will scale linearly with n.

Given the above analysis, in applications where this memory requirement may be hard to satisfy, one can directly apply Algorithm 2 to compute these matrices during the decoding. We would like to emphasize that the use of the soft-MAP decoding at the bottom layer is motivated by the fact that all projected codewords are subcodes of order-1 RM codes whose dimensions are $R_t \le m-r+2$. Given that our experiments suggest that projections with smaller R_t are favorable in the decoding process, the above matrices are often significantly smaller than the bounds analyzed here, and the

soft-MAP algorithm can be easily afforded. Nevertheless, one may extend the lower-complexity fast Hadamard transform (FHT) decoder of order-1 RM codes to subcodes of order-1 RM codes, and then apply the extended FHT algorithm (instead of MAP) in the subRPA or its soft version (instead of soft-MAP) in the soft-subRPA algorithm or for training the ML model.

Appendix C
LLRs of the Information Bits

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

$$\boldsymbol{l}_{\inf}(i) \approx \max_{\boldsymbol{c}\in\mathcal{C}_i^0} \langle \boldsymbol{l}, 1-2\boldsymbol{c}\rangle \; - \; \max_{\boldsymbol{c}\in\mathcal{C}_i^1} \langle \boldsymbol{l}, 1-2\boldsymbol{c}\rangle, \tag{16}$$

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

$$\begin{aligned}\boldsymbol{l}_{\inf}(i) &\coloneqq \ln\left(\frac{\Pr(u_i=0|\boldsymbol{y})}{\Pr(u_i=1|\boldsymbol{y})}\right)\\ &\overset{(a)}{=} \ln\left(\frac{\sum_{\boldsymbol{s}\in\mathcal{C}_i^0}\exp\left(-||\boldsymbol{y}-\boldsymbol{s}||_2^2/\sigma^2\right)}{\sum_{\boldsymbol{s}\in\mathcal{C}_i^1}\exp\left(-||\boldsymbol{y}-\boldsymbol{s}||_2^2/\sigma^2\right)}\right)\\ &\overset{(b)}{\approx} \frac{1}{\sigma^2}\min_{\boldsymbol{c}\in\mathcal{C}_i^1}||\boldsymbol{y}-\boldsymbol{s}||_2^2 - \frac{1}{\sigma^2}\min_{\boldsymbol{c}\in\mathcal{C}_i^0}||\boldsymbol{y}-\boldsymbol{s}||_2^2,\end{aligned} \tag{17}$$

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 $\boldsymbol{s}$'s have the same norm, we obtain (16).

Appendix D
Proof of Proposition 2

It is well known that the decoding complexity of the full-projection RPA-like decoding of an $\mathcal{RM}(m, r)$ code is $\mathcal{O}(n^r \log n)$ [14]. Similarly, a proof by induction can show that the decoding complexity of our algorithms for a subcode of an $\mathcal{RM}(m, r)$ code, $r > 1$, is $\mathcal{O}(n^{r-1}\mathcal{C}(m - r + 1, 1))$, where $\mathcal{C}(m', 1)$ stands for the complexity of decoding a subcode of an $\mathcal{RM}(m', 1)$ code. We note that (proof by induction) the above complexity reduces to $\mathcal{O}((\beta n)^{r-1}\mathcal{C}(m - r + 1, 1))$ for pruned-projection decoding with a pruning factor $\beta < 1$.

We first note that, assuming (soft-) MAP at the bottom layer, $\mathcal{C}(m-r+1, 1)$ can be characterized as $\mathcal{O}(n_1 2^{k_1})$, where $n_1 = 2^{m-r+1}$ is the code length and $k_1 = m - r + 2$ is the code dimension in the bottom layer. Therefore, $\mathcal{C}(m - r + 1, 1) = \mathcal{O}(2^{m-r+1}2^{m-r+2}) = \mathcal{O}(n^2/2^{2r-3})$. This complete the proof of the first part, i.e., the $\mathcal{O}(n^{r+1})$ complexity for full-projection decoding.

Next, as discussed in Appendix B, the pruning factor β can be essentially $\mathcal{O}(1/n)$ so that the number of projections in each layer, i.e., $\mathcal{O}(\beta n)$, becomes a constant. Then, $(\beta n)^{r-1} = \mathcal{O}(1)$ (though with a large constant) and the overall complexity reduces to $\mathcal{O}(\mathcal{C}(m-r+1, 1))$. This then complete the proof of the second part, i.e., $\mathcal{O}(n^2)$ complexity for pruned-projection decoding with pruning factor $\beta = \mathcal{O}(1/n)$.

Finally, as empirically observed in Section V, in most cases the selected projections by our ML training scheme have very small (nearly the smallest) ranks R_t for the projected generator matrices. Therefore, the number of codewords $2^{k_1} = 2^{R_t}$ may be upper bounded by a constant. This then reduces the complexity to $\mathcal{O}(n)$ if $2^{R_t} = \mathcal{O}(1)$, $\forall t \in [T]$, in addition to $\beta = \mathcal{O}(1/n)$.

We would like to emphasize that the complexity analysis above my require some large constants (modeled by $\mathcal{O}(1)$). Therefore, even if the complexity can linearly scale with n, the involved constants may be large. However, a major advantage of our decoding algorithms is the reduction in the latency (e.g., compared to polar codes) as all the branches involved in the decoding tree (see, e.g., Fig. 1) can be executed in parallel. We refer the readers to [14] for additional discussions on latency aspects of RPA-like decoding of RM codes.

References

[1] 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," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2021, pp. 1088–1093.

[2] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Trans. IRE Prof. Group Inf. Theory*, vol. 4, no. 4, pp. 38–49, 1954.

[3] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection," *Trans. IRE Prof. Group Electron. Comput.*, vol. EC-3, no. 3, pp. 6–12, Sep. 1954.

[4] 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, Jul. 2009.

[5] 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, May 2012.

[6] H. Hassani, S. Kudekar, O. Ordentlich, Y. Polyanskiy, and R. Urbanke, "Almost optimal scaling of Reed-Muller codes on BEC and BSC channels," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2018, pp. 311–315.

[7] G. Reeves and H. D. Pfister, "Reed-Muller codes achieve capacity on BMS channels," 2021, *arXiv:2110.14631*.

[8] I. Dumer, "Recursive decoding and its performance for low-rate Reed-Muller codes," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 811–823, May 2004.

[9] I. Dumer, "Soft-decision decoding of Reed-Muller codes: A simplified algorithm," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 954–963, Mar. 2006.

[10] 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, Mar. 2006.

[11] B. Sakkour, "Decoding of second order Reed-Muller codes with a large number of errors," in *Proc. IEEE Inf. Theory Workshop*, 2005, pp. 176–178.

[12] 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, Apr. 2017.

[13] E. Santi, C. Hager, and H. D. Pfister, "Decoding Reed-Muller codes using minimum-weight parity checks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2018, pp. 1296–1300.
[14] M. Ye and E. Abbe, "Recursive projection-aggregation decoding of Reed-Muller codes," *IEEE Trans. Inf. Theory*, vol. 66, no. 8, pp. 4948–4965, Aug. 2020.
[15] D. Fathollahi, N. Farsad, S. A. Hashemi, and M. Mondelli, "Sparse multi-decoder recursive projection aggregation for Reed-Muller codes," 2020, *arXiv:2011.12882*.
[16] M. V. Jamali, M. Fereydounian, H. Mahdavifar, and H. Hassani, "Low-complexity decoding of a class of Reed-Muller subcodes for low-capacity channels," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2021, pp. 1–6.
[17] M. Fereydounian, M. V. Jamali, H. Hassani, and H. Mahdavifar, "Channel coding at low capacity," in *Proc. IEEE Inf. Theory Workshop (ITW)*, 2019, pp. 1–5.
[18] A. V. Makkuva, X. Liu, M. V. Jamali, H. Mahdavifar, S. Oh, and P. Viswanath, "KO codes: Inventing nonlinear encoding and decoding for reliable wireless communication via deep-learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 7368–7378.
[19] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.
[20] T. Gruber, S. Cammerer, J. Hoydis, and S. T. Brink, "On deep learning-based channel decoding," in *Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, 2017, pp. 1–6.
[21] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019, pp. 2758–2768.
[22] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 194–206, May 2020.
[23] H. Kim, S. Oh, and P. Viswanath, "Physical layer communication via deep learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 5–18, May 2020.
[24] T. Akyildiz, R. Ku, N. Harder, N. Ebrahimi, and H. Mahdavifar, "ML-aided collision recovery for UHF-RFID systems," in *Proc. IEEE Int. Conf. RFID*, 2022, pp. 41–46.
[25] M. V. Jamali, H. Saber, H. Hatami, and J. H. Bae, "ProductAE: Toward training larger channel codes based on neural product codes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2021, pp. 1–6.
[26] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.
[27] K. Lee and M. E. O'Sullivan, "Algebraic soft-decision decoding of hermitian codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 6, pp. 2587–2600, Jun. 2010.
[28] Y. Wan, L. Chen, and F. Zhang, "Algebraic soft decoding of elliptic codes," *IEEE Trans. Commun.*, vol. 70, no. 3, pp. 1522–1534, Mar. 2022.
[29] A. Vardy and Y. Be'ery, "Maximum-likelihood soft decision decoding of BCH codes," *IEEE Trans. Inf. Theory*, vol. 40, no. 2, pp. 546–554, Mar. 1994.
[30] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*. vol. 16. Amsterdam, The Netherlands: Elsevier, 1977.
[31] A. J. Salomon and O. Amrani, "Augmented product codes and lattices: Reed-Muller codes and Barnes-wall lattices," *IEEE Trans. Inf. Theory*, vol. 51, no. 11, pp. 3918–3930, Nov. 2005.
[32] Y. Xie et al., "Differentiable top-k with optimal transport," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 20520–20531.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[34] "BCEWithLogitsLoss." Accessed: Jan. 26, 2021. [Online]. Available: https://pytorch.org/docs/stable/generated/torch. nn.BCEWithLogitsLoss. html
[35] M. V. Jamali, H. Saber, H. Hatami, and J. H. Bae, "ProductAE: Toward deep learning driven error-correction codes of large dimensions," 2023, *arXiv:2303.16424*.
[36] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.

Mohammad Vahid Jamali (Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Michigan, Ann Arbor, in 2022. He is currently a Senior Research Engineer at Samsung Semiconductor, Inc., San Diego, CA, USA. His general areas of research include coding and information theory, machine learning, and wireless communications.

Xiyang Liu received the B.Eng. degree in electrical engineering from Shanghai Jiao Tong University and the M.S. degree from the University of Illinois at Urbana–Champaign. He is currently pursuing the Ph.D. degree with the CSE Department, University of Washington. His research interests include deep learning, robust statistics, and differential privacy.

Ashok Vardhan Makkuva received the bachelor's degree in EE with a minor in mathematics from IIT Bombay in 2015, and the master's and Ph.D. degrees in ECE from the University of Illinois at Urbana–Champaign in 2017 and August 2022, respectively. He is a Postdoctoral Associate with EPFL. His current research interests are in theoretical and algorithmic aspects of machine learning, information theory, and coding. He is a recipient of the Best Paper Award from ACM MobiHoc 2019. He is also a recipient of several graduate student awards and fellowships, including Joan and Lalit Bahl Fellowship (twice), Sundaram Seshu International Student Fellowship, and is a finalist for the Qualcomm Innovation Fellowship 2018.

Hessam Mahdavifar (Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2007, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of California San Diego, La Jolla, in 2009, and 2012, respectively.

He is an Associate Professor with the Department of Electrical Engineering and Computer Science, University of Michigan Ann Arbor. He was a Staff Research Engineer with Samsung U.S. Research and Development, San Diego, USA, from 2012 to 2016. His main area of research is coding and information theory with applications to wireless communications, storage systems, security, and privacy. He received the NSF Career Award in 2020. He also received Best Paper Award in 2015 IEEE International Conference on RFID and the 2013 Samsung Best Paper Award. He also received two Silver Medals at the International Mathematical Olympiad in 2002 and 2003 and two Gold Medals at Iran National Mathematical Olympiad in 2001 and 2002.

Sewoong Oh received the Ph.D. degree from the Department of Electrical Engineering, Stanford University in 2011, under the supervision of A. Montanari. He is an Associate Professor with the Paul G. Allen School of Computer Science and Engineering, University of Washington. Previous to joining the University of Washington in 2019, he has been an Assistant Professor with the Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana–Champaign since 2012. Following his Ph.D., he worked as a Postdoctoral Researcher with the Laboratory for Information and Decision Systems, MIT, under the supervision of D. Shah. His research interest is in foundations of machine learning in topics including differential privacy, secure and robust machine learning, and federated learning. He was co-awarded the ACM SIGMETRICS Best Paper Award in 2015, the NSF CAREER Award in 2016, the ACM SIGMETRICS Rising Star Award in 2017, and the GOOGLE Faculty Research Awards in 2017 and 2020.

Pramod Viswanath is a Forrest G. Hamrick Professor of Engineering with Princeton University.