Universal Maximum Likelihood (List) Decoding via Fast Vector-Matrix Multiplication

Hoang Ly, and Emina Soljanin
Department of Electrical & Computer Engineering, Rutgers University
E-mail: {mh.ly;emina.soljanin}@rutgers.edu

***Abstract*—Maximum-likelihood (ML) decoding for arbitrary block codes remains fundamentally hard, with worst-case time complexity—measured by the total number of multiplications—being no better than straightforward exhaustive search, which requires $q^k n$ operations for an $[n,k]_q$ code. This paper introduces a simple, code-agnostic framework that reduces the worst-case complexity by a factor of n, down to q^k operations, a highly desirable reduction in practice. The result holds for both linear and nonlinear block codes over general memoryless channels and under both hard-decision and soft-decision decoding. It naturally extends to intersymbol-interference (ISI) channels and ML list decoding with only a negligible increase in complexity. Our core insight is that, upon receipt of each sequence at the receiver, the conditional probability of that sequence for each codeword in the codebook (i.e., the *likelihood*) can be expressed as the inner product of two carefully constructed vectors—the first depending on the received sequence, and the second on that codeword itself. As a result, evaluating the likelihoods for all codewords in the codebook reduces to a single vector-matrix multiplication, and ML decoding (MLD) becomes the simple task of picking the maximum entry in the resulting vector. The only non-trivial cost lies in the vector-matrix product. However, our matrix construction allows the use of the Mailman algorithm to reduce this cost. This time reduction is achieved at the cost of high space complexity, requiring $\mathcal{O}(q^{k+1}n)$ space to store the pre-computed codebook matrix.**

I. Introduction

Maximum Likelihood decoding is a central yet computationally challenging problem in coding theory. It was proven to be NP-complete for general linear codes by Berlekamp *et al.* in 1978 [1], [2], and this hardness extends even to highly structured code families like Reed–Solomon codes [3]. The straightforward brute-force solution, known as exhaustive search decoding (ESD), provides a simple complexity baseline for ML decoding.

In a typical setting of binary coding over a discrete memoryless channel (DMC), the sender transmits a codeword from a binary code $\mathscr{C}[n,k]_2$ with 2^k codewords, and the receiver observes a possibly corrupted sequence $\boldsymbol{y} \in \mathbb{F}_2^n$. ESD identifies the ML codeword by computing the likelihood $P(\boldsymbol{y} \mid \boldsymbol{c})$ for all $\boldsymbol{c} \in \mathscr{C}$ and selecting the maximizer. This procedure requires n operations per codeword, leading to a *time complexity* of $\mathcal{O}(2^k n)$. Although parallel hardware can evaluate many likelihoods simultaneously and thereby reduce wall-clock time, the overall number of multiplications—and hence the fundamental time complexity–remains unchanged. For this reason, complexity comparisons are conventionally made in terms of time complexity, independent of parallelization. Alongside time complexity, one also considers *space complexity*, which measures the amount of memory required by an algorithm as a function of the input size. In the case of ESD, the algorithm evaluates one codeword at a time and therefore needs only $\mathcal{O}(n)$ memory to store this codeword, making it space-efficient despite its exponential time complexity.

When the noise model is known, the likelihood measure on memoryless channels can be equivalently expressed as a distance metric—for instance, Hamming distance for memoryless symmetric channels with bit-flip noise, and Euclidean distance for additive white Gaussian noise (AWGN) channels. For example, in a binary symmetric channel (BSC), ML decoding reduces to minimum-distance decoding: the most likely codeword is the one that is closest in Hamming distance to the received sequence. For linear codes, a refinement is possible. When the code rate is $R = k/n \leq 1/2$, ESD involves checking all 2^k codewords. When $R > 1/2$, it is more efficient to consider the 2^{n-k} possible *syndromes* and perform *syndrome decoding*, which is also a minimum distance decoder and thus ML. In summary, the complexity of ESD for binary linear codes is well known to be $\mathcal{O}(2^{\min\{k,\, n-k\}} n)$ [4]. We emphasize that this refinement applies only to linear codes, since syndrome decoding is not defined for nonlinear codes. A key characteristic of the ESD is that it is *universal*[1] (i.e., *code-agnostic*) as it does not exploit any algebraic structure of the code, and thus its performance depends only on the code size rather than its specific construction.

A. Brief Historical Prospective

Due to the prohibitive complexity of ESD (linear in block length n and exponential in code dimension k), significant effort has been invested in developing more efficient ML or near-ML decoders. We provide a broad classification into a few main categories listed in Table I.

An early approach, introduced by Wolf in his seminal 1978 paper [9], demonstrated that ML decoding of any *linear* block code can be carried out using the Viterbi algorithm [10]—(originally devised for convolutional codes) on a trellis with at most 2^{n-k} states and 2^k distinct paths of depth n. This trellis-based algorithm yields a time complexity of

[1] In Information theory, the term *universal decoding* typically refers to decoding without explicit knowledge of the channel law [5], [6]. In this work, however, we use *universal* to mean that the decoder is not tied to any specific codebook, consistent with its usage in, for example, [7]. In any case, the intended meaning should be clear from the context [8].

$\mathcal{O}(2^{\min\{k,\,n-k\}}n)$, which essentially matches that of ESD, and a space complexity of at most $\mathcal{O}(2^{n-k}n)$. The algorithm's importance lies in placing ML decoding for general linear codes under a trellis framework, enabling efficient and hardware-friendly implementations, and revealing that some codes (e.g., convolutional codes) admit compact trellis representations where Viterbi decoding is significantly more efficient. The Wolf's perspective led to the recognition of *trellis complexity* as a fundamental code parameter, alongside length, dimension, and minimum distance [11], [12]. Extending this framework, Forney showed that the Viterbi algorithm can also be applied to ML decoding of linear block codes transmitted over L-tap ISI channels [13], where L denotes the channel memory, i.e., the number of past symbols that affect each received symbol. In this setting, the decoding complexity scales proportionally to 2^L. Notably, this remains the only known approach to exact ML decoding for ISI channels. The well-known BCJR algorithm [14] performs symbol-optimal detection, minimizing the symbol error rate rather than the word error rate; thus, it is not a sequence ML decoding algorithm; in fact, the optimally decoded output need not be a valid codeword.

Another approach reformulates the ML decoding problem over a discrete memoryless channel symmetric channel (NP-hard in general) as an integer linear program (ILP), whose optimal solution corresponds to the ML codeword; see [15] and related subsequent works. Relaxing the integrality constraints yields a linear program (LP) that can be solved in *polynomial time*, for instance by the Ellipsoid algorithm [16]. Under certain conditions, the LP relaxation is *tight*, meaning its solution coincides with that of the original ILP and hence produces the ML codeword in polynomial time. When these conditions fail, however, the LP solution is typically fractional and does not correspond to any valid codeword. In short, this method does not guarantee recovery of the ML codeword in general.

Practical coding standards use particular decoders co-designed with code-specific decoding methods to achieve an ML decoding performance in polynomial time for a restricted class of codes or channels. Prominent examples include Bose–Chaudhuri–Hocquenghem (BCH) codes decoded by the Berlekamp–Massey algorithm [17], decoding of expander codes over the BSC [18], and decoding of first-order Reed-Muller codes [19]. Recently, code-agnostic approaches to ML decoding have attracted significant attention, as they enable universal applicability across arbitrary codes and support practical implementations in reliable communication settings, where the SNR is typically high and moderate-to-high-rate codes are employed. This line of research was initiated by the remarkable work of Duffy et al. [20], who introduced the method of ML decoding via *noise guessing*, termed GRAND. This framework addresses discrete memoryless additive-noise channels, where the received sequence equals the transmitted codeword plus noise (modulo the alphabet). Because the noise uniquely determines the transmitted codeword given the received sequence, decoding reduces to testing (or, *guessing*) candidate noise vectors ranked by likelihood. Each is subtracted from the received sequence and checked for codebook membership; the first valid residual yields the ML codeword. The computational cost is governed by the number of guesses required before termination. Although an exact complexity is unknown, it was shown that in the asymptotic regime $n \to \infty$, the expected number of guesses scales as $2^{\,n\min\{H_{1/2},\,1-\frac{k}{n}\}}$, where H_α denotes the Rényi entropy of order α of the noise (with H_1 being the Shannon entropy). As a Viterbi decoding approach for linear block codes, this method offers substantially lower complexity than ESD in the high-SNR regime (i.e., when the noise entropy is small) and is particularly effective for high-rate, short block codes. Its advantage, however, vanishes as the SNR decreases, the code rate is reduced, and especially when the channel is non-additive.

Several techniques have been proposed to reduce the complexity of ML decoding in the high-SNR regime, typically at the cost of a slight increase in error probability. Such methods are therefore commonly referred to as *approximate* ML decoding, or *near-optimum decoding*. While asymptotically optimal at high SNR, these methods perform worse than exact ML decoding at low SNR or for practical block lengths, and the gap in error performance compared to MLD increases with the code dimension. Nevertheless, the worst-case complexity (e.g., in low-SNR conditions) remains essentially the same as that of ESD; see, for example, [21]–[23].

As a related note, extensions of ML decoding to list decoding, soft-decision decoding, and erasure decoding have been widely studied. For example, Elias's early work on list decoding (also called as list-ℓ decoding, where ℓ denotes the list size) [24] showed that allowing the decoder to output multiple candidates (i.e., allow some level of *ambiguity*) can reduce error probability, while erasure decoding provides a simple way to incorporate partial reliability information. Soft-decision decoding, particularly with ML-based algorithms, has since become the standard in practice [25]. However, each of these approaches is tailored to a specific setting, and no existing method provides a unified, code-agnostic framework that applies simultaneously to ML hard-decision, soft-decision, erasure, and list decoding.

B. The Proposed Algorithm and its Relation to the Prior Work

This paper introduces a code-agnostic framework that, for the first time, breaks the long-standing $\mathcal{O}(2^k n)$ worst-case complexity barrier of ESD. The key idea is to express the likelihood of each codeword as the inner product of two vectors: one derived from the received sequence, and the other representing the codeword itself. This allows us to represent the entire codebook as a single, large, binary matrix $\mathbf{M}$. Consequently, ML decoding is reduced to a single vector–matrix multiplication, followed by selecting the maximum entry of the resulting vector.

The efficiency of the proposed approach stems from exploiting the discrete structure of $\mathbf{M}$. This performance gain parallels the classical distinction between sorting arbitrary real numbers and sorting integers. In general, sorting n ar-

bitrary items requires $\Omega(n \log n)$ comparisons (i.e., grows on the order of $n \log n$), whereas algorithms such as Counting Sort—particularly efficient when the range of possible input values is small relative to the number of elements—leverage the finite alphabet of integers to achieve linear $\mathcal{O}(n + k)$ complexity [26], where k denotes the range size of the integer-valued inputs. Likewise, rather than treating the entries of $\mathbf{M}$ as arbitrary reals, we exploit its binary structure via the Mailman algorithm [27], which computes the product of an $m \times n$ binary matrix $\mathbf{A}$ with a real vector $\boldsymbol{x} \in \mathbb{R}^n$ in $\mathcal{O}(mn/\log \max\{m, n\})$ time—improving upon the straightforward $\mathcal{O}(mn)$ complexity for general matrices. This contrasts with the classical result of Winograd [28], which establishes that general vector-matrix multiplication requires $\Omega(mn)$ operations. Employing this algorithm reduces the total number of operations to $\mathcal{O}(2^k)$—an n-fold improvement over ESD. For the special case of linear codes over the BSC, the framework further adapts to syndrome decoding with a complexity of $\mathcal{O}(2^{\min\{k, n-k\}})$, again improving upon the standard $\mathcal{O}(n \cdot 2^{\min\{k, n-k\}})$ complexity of trellis-based ML decoders. Since the time complexity is always proportional to the number of codewords (or syndromes, whichever is smaller), no further substantial reduction is possible: identifying the most likely codeword necessarily requires considering each candidate at least once. The main drawback is memory usage: the method requires storing a matrix encoding all codewords in the codebook, which incurs a space complexity of $\mathcal{O}(2^{k+1}n)$. Nevertheless, the framework is universal: it applies to both linear and nonlinear block codes, and supports hard- and soft-decision decoding over general memoryless channels as well as intersymbol-interference (ISI) channels—the most widely studied class of channels with memory [29].

The idea of representing codeword likelihoods as inner products can be traced back, in an early form, to the work of Conway and Sloane in 1986 [30], who studied specific channels with binary codes and showed that, in those cases, ML decoding is equivalent to finding the codeword closest in Euclidean distance. By contrast, our method makes no assumptions about the noise model and is not a distance-based decoder; it can thus be viewed as a systematic and efficiently implementable generalization of their approach.

Unlike all previously known ML decoding methods, the proposed decoder universally achieves a complexity of $\mathcal{O}(2^k)$, which is independent of the block length n when the code rate k/n is assumed to be fixed. This complexity is also independent of the SNR level, distinguishing it from approaches like GRAND [20]. Moreover, the algorithm does not depend on the specific received sequence, unlike the algebraic ML decoder of [31]. The complexity of our proposed method is smaller by a logarithmic factor compared to the best known trellis-based ML decoding via the Viterbi algorithm. This makes the method particularly attractive for low-SNR channels—a relatively uncharted regime—where reliable transmission demands low-rate codes with large minimum distance. In this setting, approximate ML decoding algorithms fail to deliver satisfactory performance, and exact ML decoders cannot achieve lower complexity than ESD. Furthermore, when extended to list decoding, the algorithm's complexity increases only logarithmically with the list size, in contrast to other approaches such as SOGRAND [32] and List Viterbi [33], whose complexity grows linearly with the list size.

Our approach (just as ESD and Viterbi decoding) does not impose assumptions on the noise or channel model. In contrast, prior works such as [20], [34] assume additive noise, with [20] focusing on symmetric channels and [34] restricted to discrete (binary-input, multiple-output) channels. Our method achieves these advantages at the expense of increased storage, i.e., higher *space complexity*. This observation is consistent with the well-known principle of *time–space tradeoffs* in computer science, where reductions in computational time often require increased memory usage [35].

The proposed method is especially relevant for emerging applications such as the ultra-reliable low-latency communication (URLLC), a foundational requirement of 5G and future 6G networks. These systems face a fundamental tradeoff: to increase reliability, one often lengthens the block length n, which, for conventional universal decoders, directly increases decoding latency, often scaling as $\mathcal{O}\big(2^k n\big)$. In contrast, our method's $\mathcal{O}(2^k)$ complexity decouples latency from block length. There are two key advantages: 1) for a given message/information size k, designers can increase n to boost reliability without increasing decoding delay; and 2) the n-fold reduction in complexity becomes increasingly significant for large n. These benefits position our method as a compelling enabler in domains demanding both high reliability and minimal latency, such as the Tactile Internet, V2X (vehicle-to-everything) communication, and industrial automation [36]–[38].

C. Organization

The core contributions of this paper are presented in Section II. Part II-A introduces the universal decoding method for discrete memoryless channels and analyzes its complexity. Part II-B extends the framework to soft-decision decoding and list-ℓ ML decoding, and derives the associated complexities. Part II-C applies the method to erasure decoding. Part II-D focuses on linear codes over symmetric channels, showing how syndrome decoding can be used to further reduce complexity. Finally, Part II-E treats the generalization to L-tap ISI channels. We conclude the paper in Section III, where we also discuss the inherent difficulty of the underlying vector-matrix multiplication routine and, drawing on prior work—though without formal proof—argue that further substantial improvements in the worst-case complexity are unlikely. Finally, a brief overview of the Mailman algorithm is given in Appendix A.

II. Formulation

In the rest of this paper, the finite field over a prime or prime power q is denoted as $\mathbb{F}_q$. A q-ary **linear code** $\mathscr{C}$ with parameters $[n, k, d]_q$ is a k-dimensional subspace of the vector space $\mathbb{F}_q^n$ with minimum distance d. More generally, any set

TABLE I
COMPARISON OF EXACT ML DECODING APPROACHES FOR GENERAL LINEAR $[n,k]_2$ CODES.

Decoding Method	**Time Complexity**	**Space Complexity**	**Characteristics**
Exhaustive Search (ESD)	$\mathcal{O}(2^k n)$ in general $\mathcal{O}(2^{\min\{k,\,n-k\}} n)$ for linear codes	$\mathcal{O}(n)$	Code-agnostic; Works for all codes for both hard- and soft-decision decoding. Prohibitive time complexity.
Trellis-based (Viterbi) [9]	$\mathcal{O}(2^{\min\{k,\,n-k\}} n)$ for linear codes	$\mathcal{O}(2^{n-k} n)$	Code-agnostic; Works for linear codes only. Same time complexity as ESD.
LP Relaxation [15]	Polynomial in n		Fails if relaxation is not tight. Works for discrete memoryless symmetric channels only.
GRAND [20]	$\mathcal{O}(2^{\,n \min\{H_{1/2},\, 1-\frac{k}{n}\}} n)$ (Asymptotically)	$\mathcal{O}(n)$	Low complexity under high SNRs. Otherwise, same as ESD. Requires additive-noise model.
Algebraic Decoders [17]–[19]	Polynomial in n		Applicable to particular codes/channels only.
Our method. In general For linear codes	$\mathcal{O}(2^k)$ $\mathcal{O}(2^{\min\{k,\,n-k\}})$	$\mathcal{O}(2^{k+1} n)$ $\mathcal{O}(2^{\min\{k,\,n-k\}} \cdot 2n)$	Code-agnostic; Works for all codes for both hard- and soft-decision decoding. High space complexity.

of S codewords in $\mathbb{F}_q^n$ with minimum distance d forms a code, which may be non-linear. The logarithm, denoted by $\log$, is taken to base 2 unless stated otherwise.

A. Decoding Algorithm Formulation

We denote a generic discrete memoryless channel (DMC) by $P : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output alphabets, respectively. For example, in the BSC, $\mathcal{X} = \mathcal{Y} = \{0, 1\}$, while for the AWGN channel, $\mathcal{Y} = \mathbb{R}$. The channel is characterized by the conditional transition probabilities $P_{Y|X}(y \mid x)$ for each $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. While the output alphabet $\mathcal{Y}$ may be arbitrary, we assume the input alphabet $\mathcal{X}$ is discrete. This is because in digital communication, information is first encoded using a channel code defined over a finite alphabet before modulation and transmission. Without loss of generality, we index the input symbols as $\mathcal{X} = \{1, 2, \ldots, q\}$, where $q = |\mathcal{X}|$ denotes the alphabet size. A q-ary block code (not necessarily linear) $\mathscr{C}[n,k]_q \subseteq \mathcal{X}^n$ of block length n and information length k (i.e., the number of information symbols) is assumed to be shared between the sender and the receiver. When the code is linear, k is also referred to as the *dimension* of the code. Suppose a codeword $\boldsymbol{x} \in \mathscr{C}$ is transmitted over the channel, resulting in a received sequence $\boldsymbol{y} \in \mathcal{Y}^n$. Throughout this paper, we index vector components from 1 to their length, and use y_i and $\boldsymbol{y}[i]$ interchangeably to denote the i-th component of $\boldsymbol{y}$. For a vector $\boldsymbol{v}$, its transpose is denoted by $\boldsymbol{v}^\top$.

We write $P_{Y|X}^n$ to denote the product channel corresponding to n independent uses of the memoryless channel P, i.e.,

$$P_{Y|X}^n : \mathcal{X}^n \to \mathcal{Y}^n, \quad P_{Y|X}^n(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{i=1}^{n} P_{Y|X}(y_i \mid x_i).$$

For simplicity, we often write P instead of $P_{Y|X}$ when the context is clear. ML decoding over a DMC seeks the codeword $\boldsymbol{c}^* \in \mathscr{C}$ that maximizes the conditional probability of the received sequence $\boldsymbol{y} \in \mathcal{Y}^n$, namely

$$\boldsymbol{c}^* = \arg\max_{\boldsymbol{c} \in \mathscr{C}} P_{Y|X}^n(\boldsymbol{y} \mid \boldsymbol{c}) = \arg\max_{\boldsymbol{c} \in \mathscr{C}} \log \left(\prod_{i=1}^{n} P(y_i \mid c_i) \right)$$
$$= \arg\max_{\boldsymbol{c} \in \mathscr{C}} \sum_{i=1}^{n} \log P(y_i \mid c_i). \tag{1}$$

ML decoding guarantees the minimum probability of decoding error when the *a priori* probabilities of all the codewords are equal [39]. To facilitate this computation, we define the *(log) conditional probability vector* $V(\boldsymbol{y}) \in \mathbb{R}^{qn}$, which records the logarithmic transition probabilities across all possible input symbols $x_i \in \mathcal{X} = \{1, \ldots, q\}$. Specifically,

$$V(\boldsymbol{y}) = [\boldsymbol{v}_1(\boldsymbol{y}), \boldsymbol{v}_2(\boldsymbol{y}), \ldots, \boldsymbol{v}_n(\boldsymbol{y})], \tag{2}$$

where each sub-vector $\boldsymbol{v}_i(\boldsymbol{y}) \in \mathbb{R}^q$ stores the log-likelihoods of receiving y_i given every possible input symbol

$$\boldsymbol{v}_i(\boldsymbol{y}) = \big[\log P(y_i \mid c_i = 1), \log P(y_i \mid c_i = 2), \ldots \\ \log P(y_i \mid c_i = q-1), \log P(y_i \mid c_i = q)\big]. \tag{3}$$

We also define the *incidence vector* $I(\boldsymbol{c}) \in \{0,1\}^{qn}$, constructed from each codeword $\boldsymbol{c} \in \mathcal{X}^n$, as

$$I(\boldsymbol{c}) = [\boldsymbol{u}_1(\boldsymbol{c}), \boldsymbol{u}_2(\boldsymbol{c}), \ldots, \boldsymbol{u}_n(\boldsymbol{c})], \tag{4}$$

where each sub-vector $\boldsymbol{u}_i(\boldsymbol{c}) \in \{0,1\}^q$ is the one-hot encoding of c_i, that is

$$\boldsymbol{u}_i(\boldsymbol{c}) = \big[\mathbb{1}\{c_i = 1\}, \mathbb{1}\{c_i = 2\}, \ldots, \mathbb{1}\{c_i = q\}\big],$$

with $\mathbb{1}\{\cdot\}$ denoting the indicator function ($\mathbb{1}\{A\} = 1$ if event A is true, and 0 otherwise). Each $\boldsymbol{u}_i(\boldsymbol{c})$ therefore has a single 1 in the coordinate matching the symbol c_i, and 0s elsewhere. Consequently, $I(\boldsymbol{c})$ is a binary vector of length nq with exactly n ones. With this notation, MLD rule in Eq. (1) presents as

$$\boldsymbol{c}^* = \arg\max_{\boldsymbol{c} \in \mathscr{C}} \sum_{i=1}^{n} \boldsymbol{v}_i(\boldsymbol{y}) \cdot \boldsymbol{u}_i(\boldsymbol{c})^\top = \arg\max_{\boldsymbol{c} \in \mathscr{C}} V(\boldsymbol{y}) \cdot I(\boldsymbol{c})^\top,$$

meaning that the likelihood of each codeword is given by the inner product of two vectors. The first, $V(\boldsymbol{y})$, is a conditional probability vector determined solely by the received sequence $\boldsymbol{y}$ and the channel transition probabilities $P(\boldsymbol{y} \mid \boldsymbol{x})$, and is independent of the codebook (it only depends on the alphabet of the codebook). The second, $I(\boldsymbol{c})$, is a codeword-specific incidence vector that is independent of both the received sequence and the channel model.

Let $S = |\mathscr{C}|$ denote the size of the codebook, i.e., the number of codewords, and $[S] = \{1, 2, \ldots, S\}$. We denote the codewords be indexed as $\boldsymbol{c}^1, \boldsymbol{c}^2, \ldots, \boldsymbol{c}^S$. Since each codeword $\boldsymbol{c}^j$ is uniquely associated with a codeword index j, ML decoding then amounts to finding

$$i^* = \arg\max_{i \in [S]} V(\boldsymbol{y}) \cdot I(\boldsymbol{c}^i)^\top.$$

We collect the incidence vectors of all codewords as columns of a binary matrix

$$\mathbf{M} = \left[I(\boldsymbol{c}^1)^\top \;\; I(\boldsymbol{c}^2)^\top \;\; \cdots \;\; I(\boldsymbol{c}^S)^\top \right] \in \{0,1\}^{nq \times S}. \quad (5)$$

The resulting log-likelihood vector is given by

$$\mathbf{m} = V(\boldsymbol{y}) \cdot \mathbf{M} \quad \in \quad \mathbb{R}^S,$$

where

$$\mathbf{m}[i] = \log P(\boldsymbol{y} \mid \boldsymbol{c}^i), \, \forall \, i \in [S], \quad (6)$$

is the log likelihood of the codeword $\boldsymbol{c}^i$. Thus, ML decoding reduces to identifying the index of the maximum entry in a length-S, real-valued vector

$$i^* = \arg\max_{i \in [S]} \mathbf{m}[i],$$

which can be accomplished using a trivial linear-time scan: by iterating through the vector once and updating a register that stores the current maximum value. Since the length of the resulting vector $\mathbf{m}$ is S, this step has a computational complexity of $\mathcal{O}(S)$. Before analyzing the overall decoding complexity, we summarize the procedure in Box 1 and exemplify it in Example 1.

Box 1: ML decoding via Vector–Matrix Multiplication

Pre-processing: For a code $\mathscr{C} = \{\boldsymbol{c}^1, \boldsymbol{c}^2, \ldots, \boldsymbol{c}^S\}$, construct the binary matrix $\mathbf{M}$ as defined in Eq. (5).

- **Step 1: Construct the conditional probability vector.** Given a received sequence $\boldsymbol{y}$, build the conditional probability vector $V(\boldsymbol{y})$ according to Eq. (2) and (3).
- **Step 2: Perform vector–matrix multiplication.** Compute the score vector $\mathbf{m} = V(\boldsymbol{y}) \cdot \mathbf{M} \in \mathbb{R}^S$.
- **Step 3: Identify the maximum-likelihood codeword.** Let $i^* = \arg\max_{i \in [S]} \mathbf{m}[i]$. Then the ML codeword is
$$\boldsymbol{c}^* = \boldsymbol{c}^{i^*}.$$

Example 1. *Consider a BSC channel with bit flip probability p_e in which the non-linear binary code $\mathscr{C} = \{001, 010, 100, 111\}$ is used. Let $a := \log P(0 \mid 0) = \log(1 - p_e)$ and $b := \log P(0 \mid 1) = \log p_e$. Upon receiving a received sequence, say $\boldsymbol{y} = 000$, we construct the conditional probability vector as*

$$V(\boldsymbol{y}) = [a, \, b, \, a, \, b, \, a, \, b] \in \mathbb{R}^6.$$

The codebook matrix $\mathbf{M}$ and the resulting score vector $\mathbf{m} = V(\boldsymbol{y}) \cdot \mathbf{M}$ are

$$\mathbf{M} = \begin{pmatrix} | & | & | & | \\ I(\boldsymbol{c}^1)^\top & I(\boldsymbol{c}^2)^\top & I(\boldsymbol{c}^3)^\top & I(\boldsymbol{c}^4)^\top \\ | & | & | & | \end{pmatrix} = \left(\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right),$$

and

$$\mathbf{m} = \big(2a + b, \; 2a + b, \; 2a + b, \; 3b\big).$$

The ML decision corresponds to the maximum entry in $\mathbf{m}$. When $p_e < 0.5$, the first three entries, corresponding to the codewords all at Hamming distance 1 from $\boldsymbol{y}$, are maximal. The algorithm requires storing at the receiver a binary matrix $\mathbf{M}$ of size 6×4. □

Complexity Analysis. We now analyze the time complexity and space complexity of the algorithm.

Time complexity. The overall complexity is obtained by summing the cost of the three following steps

- **Step 1** requires constructing the conditional probability vector $V(\boldsymbol{y}) \in \mathbb{R}^{qn}$, which has complexity $\mathcal{O}(nq)$.
- **Step 2** involves a vector–matrix multiplication between a real-valued vector $V(\boldsymbol{y})$ of length nq and a binary matrix $\mathbf{M}$ of size $nq \times S$. The matrix $\mathbf{M}$ by our construction has a specific structure: it is not only binary but also highly sparse, containing exactly n ones per column. Its nature allows this product to be computed efficiently using the Mailman algorithm [27] (see Appendix A for details), with complexity
$$\mathcal{O}\left(\frac{nq \cdot S}{\log\left(\max\{nq, S\}\right)}\right). \quad (7)$$
For a q-ary linear code $\mathscr{C}$ of dimension k with $S = q^k$ codewords, and assuming $\max\{nq, q^k\} = q^k$, the complexity simplifies to
$$\mathcal{O}\left(\frac{nq \cdot q^k}{k \log q}\right) = \mathcal{O}\left(\frac{1}{R} \cdot \frac{q}{\log q} \cdot q^k\right) = \mathcal{O}(q^k),$$
where $R = k/n$ denotes fixed code rate of the particular code under consideration. The term $q/\log q$ is also treated as a constant for any practical code, e.g., for binary codes, $q/\log_2 q = 2$.
- **Step 3** finds the maximum entry in the score vector $\mathbf{m}$ of length S via a linear-time scan, with complexity $\mathcal{O}(S) = \mathcal{O}(q^k)$ for a q-ary linear $[n, k]_q$ code. Note that since the time complexity is already proportional to the number of codewords S, no further substantial reduction is expected, as identifying the most likely codeword requires examining each candidate at least once.

Hence, the total decoding time complexity is

$$\mathcal{O}(nq) + \mathcal{O}(q^k) + \mathcal{O}(q^k) = \mathcal{O}(q^k),$$

since, for practical block codes, the number of information symbols k increases roughly in proportion to the block length n, making the term nq negligible compared to q^k.

Space complexity. Our algorithm requires constructing and storing a matrix $\mathbf{M}$ that encodes all codewords in the code $\mathscr{C}$, i.e., the entire *universe*. This matrix has dimensions $nq \times S$. For a q-ary linear code of dimension k with $S = q^k$ codewords, this results in a space complexity of $\mathcal{O}(q^{k+1}n)$.

Remark 1. *Step 2 involves first constructing the matrix $\mathbf{M}$ from all codewords in the code $\mathscr{C}$, followed by its factorization*

$$\mathbf{M} = UP$$

to enable the use of the Mailman algorithm (see Appendix A). Both operations can be performed offline at the receiver and thus are treated as one-time pre-processing steps. Their complexities are amortized and excluded from the complexity analysis. This treatment aligns with standard practices in prior works, such as [34], and is analogous to the pre-processing steps required to construct the trellis in the Viterbi algorithm [9].

Remark 2. *For a $[n, k]_q$ code, the proposed decoding method has complexity $\mathcal{O}(q^k)$, which depends only on the alphabet size q and code dimension k, but not on the block length n. This complexity is smaller by a factor of n (i.e., a logarithmic factor gained through the denominator in Eq. (7)) compared to ESD, as well as other universal decoding methods; see Section I for details, and offers a significant gain for codes with long block length.*

B. Extension to Soft Decoding and List Decoding

We now demonstrate that the proposed method extends naturally to both (maximum likelihood) soft decoding and list decoding.

Soft Decoding: Observe that the output alphabet $\mathcal{Y}$ and the (conditional) transition probabilities can be arbitrary. Particularly, if the received vector $\boldsymbol{y}$ is considered *prior to demodulation*—for instance, as the real-valued output of a matched filter—then the output alphabet becomes continuous, i.e., $\mathcal{Y} = \mathbb{R}$. This scenario corresponds to *soft decoding*, where the channel transition probability $P_{Y|X}$ becomes a conditional probability *density function* $p_{Y|X}$. It is well known—see, for example, [39, Ch. 4]—that ML decoding in this setting is

$$\begin{aligned}\boldsymbol{c}^* &= \arg\max_{\boldsymbol{c}\in\mathscr{C}} p_{Y|X}(\boldsymbol{y} \mid \boldsymbol{c}) = \arg\max_{\boldsymbol{c}\in\mathscr{C}} \log\left(\prod_{i=1}^{n} p(y_i \mid c_i)\right) \\ &= \arg\max_{\boldsymbol{c}\in\mathscr{C}} \sum_{i=1}^{n} \log p(y_i \mid c_i).\end{aligned}$$

that is, the task of finding the codeword $\boldsymbol{c}$ that maximizes the likelihood $p_{Y|X}(\boldsymbol{y} \mid \boldsymbol{c})$. This is structurally equivalent to the standard ML decoding formulation given in Eq. (1) and can therefore be handled using the same approach.

List Decoding: In unique ML decoding, the goal is to find the codeword $\boldsymbol{c}^* \in \mathscr{C}$ that maximizes $P_{Y|X}(\boldsymbol{y} \mid \boldsymbol{c})$. In list-$\ell$ ML decoding, where the list size ℓ is fixed, the objective is to return a list of $\ell \geq 1$ codewords whose likelihoods are the ℓ largest among all codewords in the codebook. Formally, one seeks a list $\mathcal{L}_\ell \subseteq [S]$ such that

$$\begin{aligned}\mathcal{L}_\ell = \{i_1, \ldots, i_\ell\} \text{ with } \mathbf{m}[i_j] \geq \mathbf{m}[h], \quad \forall\, h \in [S] \setminus \mathcal{L}_\ell, \\ \text{and } \mathbf{m}[i_1] \geq \mathbf{m}[i_2] \geq \cdots \geq \mathbf{m}[i_\ell].\end{aligned}$$

This reduces to the problem of selecting and sorting the ℓ largest elements from an unordered list of S values. This task can be done with complexity $\mathcal{O}(S \log \ell)$ by using, for instance, a Min-heap–based selection algorithm [40]. The time complexity grows only logarithmically with the list size ℓ.

C. Application to Erasure Channels

The proposed framework extends naturally to binary erasure channels. On such a channel, the goal is to find the unique codeword $\boldsymbol{c}^*$ that is consistent with the received sequence $\boldsymbol{y}$ on all unerased positions, a task that is guaranteed to have a unique solution if the number of erasures n_ϵ is less than the code's minimum distance d.

The key adaptation is to use a bipolar encoding scheme instead of the one-hot encoding used for the DMC. Let $\mathcal{K}$ be the set of known (i.e., unerased) positions. We define a vector based on the received sequence $\boldsymbol{y} \in \{0, 1, \epsilon\}^n$ and a vector for each codeword $\boldsymbol{c} \in \{0, 1\}^n$ as follows:

$$V(\boldsymbol{y})[i] = \begin{cases} 1, & \text{if } y_i = 1 \\ -1, & \text{if } y_i = 0 \\ 0, & \text{if } y_i = \epsilon \end{cases} \text{ and } I(\boldsymbol{c})[i] = \begin{cases} 1, & \text{if } c_i = 1 \\ -1, & \text{if } c_i = 0 \end{cases}$$

With this representation, the product $V(\boldsymbol{y})[i]\, I(\boldsymbol{c})[i]$ equals 0 only when y_i is erased; otherwise, it equals 1 if y_i and c_i agree, and -1 if they differ. The inner product then has a remarkable property. It directly computes a score that is maximized by the codeword with the fewest disagreements on the unerased positions with sequence $\boldsymbol{y}$. Let $d_{\mathcal{K}}(\boldsymbol{y}, \boldsymbol{c})$ be the number of positions in $\mathcal{K}$ where $y_i \neq c_i$, then $|\mathcal{K}| - d_{\mathcal{K}}(\boldsymbol{y}, \boldsymbol{c})$ is the number of positions that $\boldsymbol{y}$ and $\boldsymbol{c}$ match. The inner product evaluates to

$$\begin{aligned}V(\boldsymbol{y}) \cdot I(\boldsymbol{c})^\top = \\ (|\mathcal{K}| - d_{\mathcal{K}}(\boldsymbol{y}, \boldsymbol{c})) - d_{\mathcal{K}}(\boldsymbol{y}, \boldsymbol{c}) = |\mathcal{K}| - 2d_{\mathcal{K}}(\boldsymbol{y}, \boldsymbol{c}). \quad (8)\end{aligned}$$

Maximizing this score is equivalent to minimizing $d_{\mathcal{K}}(\boldsymbol{y}, \boldsymbol{c})$. The unique correct codeword $\boldsymbol{c}^*$ has $d_{\mathcal{K}}(\boldsymbol{y}, \boldsymbol{c}^*) = 0$, which yields the maximum possible score of $|\mathcal{K}| = n - n_\epsilon$.

The decoding procedure is therefore identical to the main formulation. For a code $\mathscr{C}$ of S codewords, we first precompute the codebook matrix $\mathbf{M} = [I(\boldsymbol{c}^1)^\top, \ldots, I(\boldsymbol{c}^S)^\top]$ over $\{-1, 1\}^{n \times S}$, and for a given $\boldsymbol{y}$, we compute the score vector:

$$\mathbf{m} = V(\boldsymbol{y}) \cdot \mathbf{M}.$$

The index of the decoded codeword is simply $i^* = \arg\max_i \mathbf{m}[i]$. Since the alphabet of the matrix $\mathbf{M}$ has size

2, the vector-matrix multiplication step can be accelerated by the Mailman algorithm to reduce runtime in the same way as in our DMC vector–matrix formulation. The complexity of constructing the matrix $\mathbf{M}$ is amortized. The computational complexity is therefore dominated by the complexity of the vector-matrix multiplication step:

$$\mathcal{O}(n) + \mathcal{O}\left(\frac{n \cdot S}{\log\left(\max\{n, S\}\right)}\right) = \mathcal{O}(S).$$

This approach can be generalized to q-ary erasure channels with a similar encoding strategy.

D. Application to Syndrome Decoding for Linear Codes

For the special case of a binary linear code over a BSC, ML decoding is equivalent to finding the most likely error pattern. Let $\mathscr{C}[n,k]_2$ be a binary linear code with parity-check matrix $\mathbf{H}$, and let $\mathcal{L} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_{2^{n-k}}\}$ denote the set of all 2^{n-k} distinct coset leaders. Upon receiving a vector $\boldsymbol{y} \in \mathbb{F}_2^n$, the decoder's task is to identify the coset leader $\boldsymbol{e}_j \in \mathcal{L}$ whose syndrome $\boldsymbol{b}_j = \mathbf{H}\boldsymbol{e}_j^\top$ exactly matches the received syndrome $\boldsymbol{s} = \mathbf{H}\boldsymbol{y}^\top$. We next show how our framework can be adapted to solve this exact-match problem efficiently, reducing the decoding complexity to $\mathcal{O}\big(2^{\min\{k,\, n-k\}}\big)$.

The objective is to compute a vector $\boldsymbol{d}$ of length 2^{n-k} consisting of Hamming distances, where each entry $\boldsymbol{d}[j]$ is the distance $d_H(\boldsymbol{s}, \boldsymbol{b}_j)$ between the received syndrome and the syndrome of the j-th coset leader. We then find the index j^* where this distance is zero. To express this computation as a vector-matrix product, we define a specific encoding for the syndrome bits. For any bits $a, b \in \{0,1\}$, let

$$v(a) = [1-a, a] \quad \text{and} \quad u(b) = [b, 1-b].$$

This encoding has the crucial property that the inner product $v(a) \cdot u(b)^\top = (1-a)b + a(1-b) = \mathbb{1}\{a \neq b\}$, which is 1 if the bits differ and 0 otherwise. We now construct the vector and matrix for our syndrome-decoding problem, analogous to $V(\boldsymbol{y})$ and $\mathbf{M}$ in the main formulation.

- **Syndrome Vector $V_{\text{synd}}(\boldsymbol{s})$:** Given the received syndrome $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$, we construct a real-valued vector of length $2(n-k)$ by concatenating the $v(\cdot)$ encodings of its bits:

$$V_{\text{synd}}(\boldsymbol{s}) = [v(s_1), v(s_2), \ldots, v(s_{n-k})].$$

- **Syndrome Matrix $\mathbf{M}_{\text{synd}}$:** During a one-time preprocessing, we compute the syndrome $\boldsymbol{b}_j$ for every coset leader $\boldsymbol{e}_j \in \mathcal{L}$. We then construct a binary matrix $\mathbf{M}_{\text{synd}} \in \{0,1\}^{2(n-k)\times 2^{n-k}}$ where each column j is the concatenated $u(\cdot)$ encodings of the bits of $\boldsymbol{b}_j$:

$$\mathbf{M}_{\text{synd}} = \big[U(\boldsymbol{b}_1)^\top \;\; U(\boldsymbol{b}_2)^\top \;\; \cdots \;\; U(\boldsymbol{b}_{2^{n-k}})^\top\big],$$

where

$$U(\boldsymbol{b}_j) = [u(\boldsymbol{b}_j[1]), u(\boldsymbol{b}_j[2]), \ldots, u(\boldsymbol{b}_j[n-k])] \in \mathbb{F}_2^{2(n-k)}.$$

By this construction, the vector-matrix product directly yields the vector of Hamming distances:

$$\boldsymbol{d} = V_{\text{synd}}(\boldsymbol{s}) \cdot \mathbf{M}_{\text{synd}},$$
$$\text{where } \boldsymbol{d}[j] = \sum_{i=1}^{n-k} v(s_i) \cdot u(\boldsymbol{b}_j[i])^\top = d_H(\boldsymbol{s}, \boldsymbol{b}_j).$$

The ML decoding task reduces to computing $\boldsymbol{d}$ via the Mailman algorithm and finding the index $j^* = \arg\min_j \boldsymbol{d}[j]$. The most likely error pattern is $\boldsymbol{e}_{j^*}$, and the decoded codeword is $\boldsymbol{c}^* = \boldsymbol{y} \oplus \boldsymbol{e}_{j^*}$.

Since the matrix $\mathbf{M}_{\text{synd}}$ is binary, the vector-matrix multiplication step can be accelerated by the Mailman algorithm. The complexity of constructing the syndrome matrix $\mathbf{M}_{\text{synd}}$ is amortized. The computational complexity is therefore determined by the size of the coset leader search space:

$$\mathcal{O}(2(n-k)) + \mathcal{O}\left(\frac{2(n-k)\cdot 2^{n-k}}{\log\left(\max\{2(n-k), 2^{n-k}\}\right)}\right) = \mathcal{O}(2^{n-k}).$$

Combining this with $\mathcal{O}(2^k)$—the complexity established earlier in the general case of ML decoding of q-ary codes over DMC, the overall worst-case complexity for ML decoding of linear codes on the BSC via this approach is $\mathcal{O}(2^{\min\{k,\, n-k\}})$. It is straightforward to extend the method to ML decoding of q-ary symmetric channels using q-ary linear codes, in which case the total complexity is $\mathcal{O}(q^{\min\{k,\, n-k\}})$.

E. Extension to Intersymbol-Interference (ISI) Channels

We now extend our method to an L-tap intersymbol interference (ISI) channel, where the output y_i at time i depends on the current and L previous inputs, characterized by the transition probability $P(y_i \mid x_i, x_{i-1}, \ldots, x_{i-L})$. When $L = 0$, this corresponds to the DMC considered previously.

The ML decoding rule for a received sequence $\boldsymbol{y}$ is to find the codeword $\boldsymbol{c} \in \mathscr{C}$ that maximizes

$$\begin{aligned}\boldsymbol{c}^* &= \arg\max_{\boldsymbol{c}\in\mathscr{C}} \prod_{i=1}^{n} P(y_i \mid c_i, c_{i-1}, \ldots, c_{i-L}) \\ &= \arg\max_{\boldsymbol{c}\in\mathscr{C}} \sum_{i=1}^{n} \log P(y_i \mid c_i, c_{i-1}, \ldots, c_{i-L}). \qquad (9)\end{aligned}$$

The proposed framework can be directly applied by redefining the fundamental unit of encoding. Instead of considering individual symbols from an alphabet of size q, we consider the state defined by the input tuple $(x_i, x_{i-1}, \ldots, x_{i-L})$. There are q^{L+1} such possible tuples.

The proposed framework naturally extends to this setting by redefining the fundamental encoding unit. Instead of treating individual input symbols from an alphabet of size q, we consider the composite state formed by each input tuple $(x_i, x_{i-1}, \ldots, x_{i-L})$. There are q^{L+1} possible such tuples.

Accordingly, the core vectors introduced in Section II-A are modified as follows:

$$V(\boldsymbol{y}) = \big[\, \boldsymbol{v}_1(\boldsymbol{y}),\, \boldsymbol{v}_2(\boldsymbol{y}),\, \ldots,\, \boldsymbol{v}_n(\boldsymbol{y}) \,\big],$$
$$I(\boldsymbol{c}) = \big[\, \boldsymbol{u}_1(\boldsymbol{c}),\, \boldsymbol{u}_2(\boldsymbol{c}),\, \ldots,\, \boldsymbol{u}_n(\boldsymbol{c}) \,\big].$$

1) **Conditional Probability Vector $V(\boldsymbol{y})$:** This vector lies in $\mathbb{R}^{nq^{L+1}}$. Each subvector $\boldsymbol{v}_i$ has q^{L+1} entries, storing the log-likelihood values $\log P(y_i \mid \cdot)$ for all possible input tuples of length $L+1$.
2) **Incidence Vector $I(\boldsymbol{c})$:** This vector lies in $\{0,1\}^{nq^{L+1}}$. Each subvector $\boldsymbol{u}_i$ is a one-hot encoding of the specific input tuple $(c_i, c_{i-1}, \ldots, c_{i-L})$ observed at position i of the codeword.

With these redefined vectors, the ML decoding process remains identical to the memoryless case: we construct a binary matrix $\mathbf{M} \in \{0,1\}^{nq^{L+1} \times S}$ from the incidence vectors of all codewords and compute the score vector $\mathbf{m} = V(\boldsymbol{y}) \cdot \mathbf{M}$, where S denotes the number of codewords in the codebook. The ML codeword corresponds to the maximum entry in $\mathbf{m}$.

The computational complexity of the vector-matrix multiplication step becomes

$$\mathcal{O}\left(\frac{nq^{L+1}S}{\log(\max\{nq^{L+1},\, S\})}\right).$$

For a code with $S = q^k$ codewords and assuming $nq^{L+1} \leq q^k$, the complexity simplifies to

$$\mathcal{O}\left(\frac{nq^{L+1} \cdot q^k}{k \log q}\right) = \mathcal{O}\left(\frac{1}{R} \cdot \frac{q}{\log q} \cdot q^{k+L}\right) = \mathcal{O}(q^{k+L}),$$

where $R = k/n$ is the code rate. The dominant cost is thus determined by the codebook size and channel memory length.

Following the same logic, the proposed framework for ISI channels also extends straightforwardly to maximum-likelihood list, soft, and erasure decoding.

III. Conclusion

We developed an ML decoding algorithm for arbitrary block codes with n-fold complexity reduction for an length-n code. From a practical perspective, the proposed framework is well-suited to latency-critical applications where fast and deterministic decoding is essential. In particular, ultra-reliable low-latency communication (URLLC) systems—central to 5G and beyond—require sub-millisecond decoding to support diverse delay-sensitive domains such as the Tactile Internet, industrial automation, and vehicle-to-everything (V2X) communication [36]–[38]. The proposed approach's n-fold reduction in complexity allows system designers to increase redundancy (and thus reliability) without incurring latency penalties, offering a direct advantage over iterative or trellis-based decoding. The framework's simplicity and reliance on vector-matrix multiplication also make it amenable to efficient implementation on parallel hardware such as GPUs or FPGAs, enabling practical deployment in real-time communication and storage systems where both reliability and latency are critical.

Vector–matrix multiplication is a common, basic computational routine in numerical linear algebra, optimization, machine learning, and large-scale graph analytics, among many other applications [41]. Its computational complexity remains a central problem in theoretical computer science, where even mild asymptotic improvements can have a wide impact [42]. For a general $m \times n$ matrix $\mathbf{A}$ (where $m < n$) and an n-dimensional vector $\boldsymbol{x}$, the straightforward computation of $\mathbf{A}\boldsymbol{x}$ requires $\mathcal{O}(mn)$ time. The key to achieving faster computation lies in exploiting the matrix's discrete structure. Specifically, when $\mathbf{A}$ has entries drawn from a small finite alphabet—as is the case in our construction—the Mailman algorithm can precompute and reuse partial inner products corresponding to recurring column patterns. This method reduces redundant multiplications and yields a logarithmic improvement, lowering the complexity to $\mathcal{O}(mn/\log n)$ after a one-time preprocessing step. Further refinements improve it to $\mathcal{O}(mn/\log^2 n)$ [43]. In our setting, this results in an overall runtime of $\mathcal{O}\left(\frac{q^k}{k}\right)$. These finite-alphabet acceleration techniques have also been used recently to speed up the decoding of Hidden Markov Models (HMMs), achieving a logarithmic factor improvement over the Viterbi algorithm [44]. Beyond such logarithmic gains, further improvements appear unlikely: achieving more than a polylogarithmic speedup over Viterbi has been shown to be theoretically hard [45].

Despite these advances, a significant barrier is posed by the widely believed Online Matrix–Vector (OMv) conjecture from theoretical computer science. It asserts that a truly sub-mn time per multiplication—that is, time $o(mn)$ asymptotically smaller than the product of the dimensions—is impossible for general cases. This conjecture is frequently used to establish conditional lower bounds for various computational algorithms across multiple domains [46]. Moreover, additional results show that even with unlimited preprocessing and memory, certain variants of the problem still require near-quadratic time, reinforcing the belief that polynomial improvements are unlikely [47]. For this reason, we consider it improbable that any exact, universal ML decoding method could achieve substantially lower complexity than our proposed algorithm.

Acknowledgment

This was supported in part by NSF-BSF grant FET-2120262. We thank Swapnil Saha, Michael Schleppy, and Karthik C. S. for helpful discussions.

References

[1] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Trans. on Inform. Th.*, vol. 24, no. 3, pp. 384–386, 1978.

[2] A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Trans. on Inform. Th.*, vol. 43, no. 6, pp. 1757–1766, 1997.

[3] V. Guruswami and A. Vardy, "Maximum-likelihood decoding of Reed-Solomon codes is NP-hard," *IEEE Trans. on Inform. Th.*, vol. 51, no. 7, pp. 2249–2256, 2005.

[4] J. Coffey and R. Goodman, "The complexity of inform. set decoding," *IEEE Trans. on Inform. Th.*, vol. 36, no. 5, pp. 1031–1037, 1990.

[5] H. Joudeh, "On guessing random additive noise decoding," in *IEEE Internat. Symposium on Inform. Th. (ISIT)*, 2024, pp. 1291–1296.

[6] H. K. Miyamoto and S. Yang, "On universal decoding over discrete additive channels by noise guessing," *arXiv preprint arXiv:2501.12971*, 2025. [Online]. Available: https://arxiv.org/abs/2501.12971v2

[7] A. Riaz, A. Yasar, F. Ercan, W. An, J. Ngo, K. Galligan, M. Médard, K. R. Duffy, and R. Tugce Yazicigil, "A sub-0.8-pj/bit universal soft-detection decoder using ORBGRAND," *IEEE Journal of Solid-State Circuits*, vol. 60, no. 7, pp. 2645–2659, 2025.

[8] K. Duffy, Personal communication, 2025.

[9] J. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Trans. on Inform. Th.*, vol. 24, no. 1, pp. 76–80, 1978.

[10] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. on Inform. Th.*, vol. 13, no. 2, pp. 260–269, 1967.
[11] O. Ytrehus, "On the trellis complexity of certain binary linear block codes," *IEEE Trans. on Inform. Th.*, vol. 41, no. 2, pp. 559–560, 1995.
[12] A. Kiely, S. Dolinar, R. McEliece, L. Ekroot, and W. Lin, "Trellis decoding complexity of linear block codes," *IEEE Trans. on Inform. Th.*, vol. 42, no. 6, pp. 1687–1697, 1996.
[13] G. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. on Inform. Th.*, vol. 18, no. 3, pp. 363–378, 1972.
[14] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Trans. on Inform. Th.*, vol. 20, no. 2, pp. 284–287, 1974.
[15] J. Feldman, M. Wainwright, and D. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. on Inform. Th.*, vol. 51, no. 3, pp. 954–972, 2005.
[16] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, 1st ed. Athena Scientific, 1997.
[17] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. on Inform. Th.*, vol. 15, no. 1, pp. 122–127, 1969.
[18] W. Xu and B. Hassibi, "On the complexity of exact maximum-likelihood decoding for asymptotically good low density parity check codes: A new perspective," in *2007 IEEE Inform. Th. Workshop*, 2007, pp. 150–155.
[19] K.-U. Schmidt and A. Finger, "Simple maximum-likelihood decoding of generalized first-order Reed-Muller codes," *IEEE Communications Letters*, vol. 9, no. 10, pp. 912–914, 2005.
[20] K. R. Duffy, J. Li, and M. Médard, "Capacity-achieving guessing random additive noise decoding," *IEEE Trans. on Inform. Th.*, vol. 65, no. 7, pp. 4023–4040, 2019.
[21] D. Chase, "Class of algorithms for decoding block codes with channel measurement inform." *IEEE Trans. on Inform. Th.*, vol. 18, no. 1, pp. 170–182, 1972.
[22] B. Dorsch, "A decoding algorithm for binary block codes and j-ary output channels (corresp.)," *IEEE Trans. on Inform. Th.*, vol. 20, no. 3, pp. 391–394, 1974.
[23] M. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. on Inform. Th.*, vol. 41, no. 5, pp. 1379–1396, 1995.
[24] P. Elias, "List decoding for noisy channels," in *1957 IRE WESCON Convention Record, Part 2*, Cambridge, MA, Sep. 20 1957, pp. 94–104.
[25] G. Forney and A. Vardy, "Generalized minimum-distance decoding of Euclidean-space codes and lattices," *IEEE Trans. on Inform. Th.*, vol. 42, no. 6, pp. 1992–2026, 1996.
[26] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Addison-Wesley, 1998.
[27] E. Liberty and S. W. Zucker, "The Mailman algorithm: A note on matrix–vector multiplication," *Inform. Processing Letters*, vol. 109, no. 3, pp. 179–182, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020019008002949
[28] S. Winograd, "On the number of multiplications necessary to compute certain functions," *Communications on Pure and Applied Mathematics*, vol. 23, no. 2, pp. 165–179, 1970. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160230204
[29] H. D. Pfister, "On the capacity of finite-state channels and the analysis of convolutional accumulate-m codes," Ph.D. thesis, University of California, San Diego, La Jolla, CA, 2003.
[30] J. Conway and N. Sloane, "Soft decoding techniques for codes and lattices, including the Golay code and the Leech lattice," *IEEE Trans. on Inform. Th.*, vol. 32, no. 1, pp. 41–50, 1986.
[31] T. Kaneko, T. Nishijima, H. Inazumi, and S. Hirasawa, "An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder," *IEEE Trans. on Inform. Th.*, vol. 40, no. 2, pp. 320–327, 1994.
[32] P. Yuan, M. Médard, K. Galligan, and K. R. Duffy, "Soft-output (SO) GRAND and iterative decoding to outperform LDPC codes," *IEEE Trans. on Wireless Communications*, vol. 24, no. 4, pp. 3386–3399, 2025.
[33] N. Seshadri and C.-E. Sundberg, "List Viterbi decoding algorithms with applications," *IEEE Trans. on Communications*, vol. 42, no. 234, pp. 313–323, 1994.
[34] X. Zheng and X. Ma, "A universal list decoding algorithm with application to decoding of Polar codes," *IEEE Trans. on Inform. Th.*, vol. 71, no. 2, pp. 975–995, 2025.
[35] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.
[36] T. K. Le, C. S. Liew, and K.-K. Wong, "An overview of physical layer design for ultra-reliable low-latency communication (urllc)," *IEEE Communications Surveys & Tutorials*, 2023, available online: https://www.eurecom.fr/publication/6437.
[37] A. Maghsoudnia, P. Patras, and H. Ahmadi, "Ultra-reliable low-latency in 5g: A close reality or a distant goal?" in *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks (HotNets'24)*. Ithaca, NY, USA: ACM, 2024.
[38] G. Yue, J. Zhang, and Z. Qin, "Channel coding and decoding schemes for urllc," in *Ultra-Reliable and Low-Latency Communications (URLLC) in 5G Networks*. Wiley, 2023, pp. 121–145.
[39] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York: McGraw-Hill, 2008.
[40] GeeksforGeeks. (2024) k$^{\text{th}}$ largest (or smallest) elements in an array. Accessed: 2025-07-31. [Online]. Available: https://www.geeksforgeeks.org/dsa/k-largestor-smallest-elements-in-an-array/
[41] D. Buono, F. Petrini, F. Checconi, X. Liu, X. Que, C. Long, and T.-C. Tuan, "Optimizing sparse matrix-vector multiplication for large-scale data analytics," in *Proceedings of the 2016 Internat. Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2925426.2926278
[42] E. Anand, J. van den Brand, and R. McCarty, "The structural complexity of matrix-vector multiplication," *arXiv preprint arXiv:2502.21240*, 2025, submitted 28 Feb 2025; revised 6 Mar 2025. [Online]. Available: https://arxiv.org/abs/2502.21240
[43] R. Williams, "Matrix-vector multiplication in sub-quadratic time: (some preprocessing required)," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. USA: Society for Industrial and Applied Mathematics, 2007, p. 995–1001.
[44] M. Cairo, G. Farina, and R. Rizzi, "Decoding Hidden Markov Models faster than Viterbi via online matrix-vector (max, +)-multiplication," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 1484–1490.
[45] A. Backurs and C. Tzamos, "Improving Viterbi is hard: Better runtimes imply faster clique algorithms," in *Proceedings of the 34th Internat. Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 311–321. [Online]. Available: http://proceedings.mlr.press/v70/backurs17a.html
[46] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *Proceedings of the Forty-Seventh Annual ACM Symposium on Th. of Computing*, ser. STOC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 21–30. [Online]. Available: https://doi.org/10.1145/2746539.2746609
[47] R. C. A. Gronlund and K. G. Larsen, "New unconditional hardness results for dynamic and online problems," in *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, ser. FOCS '15. USA: IEEE Computer Society, 2015, p. 1089–1107. [Online]. Available: https://doi.org/10.1109/FOCS.2015.71
[48] F. Haddadpour and V. R. Cadambe, "Codes for distributed finite alphabet matrix-vector multiplication," in *2018 IEEE Internat. Symposium on Inform. Th. (ISIT)*, 2018, pp. 1625–1629.

APPENDIX

THE MAILMAN ALGORITHM FOR FAST VECTOR-MATRIX MULTIPLICATION

The *Mailman algorithm*, introduced by Liberty and Zucker [27], is a fast vector-matrix multiplication scheme for matrices over small alphabets. It is designed to efficiently compute the product of a matrix $\mathbf{M} \in \mathcal{X}^{m \times n}$, where $\mathcal{X} \subset \mathbb{R}$ is a finite alphabet, and a real-valued vector $\boldsymbol{x} \in \mathbb{R}^n$. When the number of rows m is not too large relative to the number of columns n, and the matrix $\mathbf{M}$ has repeated column patterns, the algorithm achieves significant speedup over the straightforward $O(mn)$ complexity by reducing redundancy.

This condition makes the method particularly well-suited for structured matrix computing applications, as in distributed computing [48], where $\mathbf{M}$ has a small alphabet but large sizes.

Problem Setup

Let $\mathbf{M} \in \mathcal{X}^{m\times n}$ be a matrix with entries from a finite alphabet $\mathcal{X} \subset \mathbb{R}$, and let $\boldsymbol{x} \in \mathbb{R}^n$ be an input vector. The goal is to compute the product $\mathbf{M}\boldsymbol{x}$ efficiently, especially when $\mathbf{M}$ has low alphabet cardinality (e.g., when $\mathbf{M}$ is binary).

Core Idea for $m = \log_2 n$

In the special case where $m = \log_2 n$ and $\mathcal{X} = \{0, 1\}$, every possible binary column vector of length m appears exactly once in $\mathbf{M}$, or can be indexed via a universal matrix $U_n \in \{0, 1\}^{m\times n}$ whose columns enumerate all pairwise distinct binary strings of length m. Any such matrix $\mathbf{M}$ can then be written as

$$\mathbf{M} = U_n P,$$

where $P \in \{0, 1\}^{n\times n}$ is a sparse *correspondence matrix* where $P(i, j) = 1$ if the j-th column of $\mathbf{M}$ matches the i-th column of U_n. Then the vector-matrix product reduces to

$$\mathbf{M}\boldsymbol{x} = U_n(P\boldsymbol{x}).$$

Since P is a sparse matrix containing exactly one nonzero entry per column, computing $P\boldsymbol{x}$ takes $O(n)$ time. The key remaining step is evaluating $U_n\boldsymbol{z}$ for $\boldsymbol{z} \in \mathbb{R}^n$.

Recursive Evaluation of $U_n\boldsymbol{z}$

Let $T(n)$ denote the number of scalar additions and multiplications needed to compute $U_n\boldsymbol{z}$ for $\boldsymbol{z} \in \mathbb{R}^n$. The recursive structure of U_n is

$$U_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad U_n = \begin{bmatrix} \mathbf{0}_{n/2}^\top & \mathbf{1}_{n/2}^\top \\ U_{n/2}^\top & U_{n/2}^\top \end{bmatrix}.$$

We thus obtain the following recurrence:

$$T(n) = T(n/2) + 2n, \quad T(2) = 2 \quad \Rightarrow \quad T(n) \le 4n.$$

Hence, $U_n\boldsymbol{z}$ can be computed in $\mathcal{O}(n)$ time.

General Case: Arbitrary m

When $m \neq \log_2 n$, we partition $\mathbf{M}$ row-wise into $\lceil m/\log_2 n \rceil$ submatrices, each of height at most $\log_2 n$. That is,

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}^{(1)}, \mathbf{M}^{(2)}, \ldots, \mathbf{M}^{(r)} \end{bmatrix},$$

where each $\mathbf{M}^{(i)} \in \mathcal{X}^{m_i\times n}$, $m_i \le \log_2 n$. We apply the Mailman method to each $\mathbf{M}^{(i)}\boldsymbol{x}$ separately, each taking $O(n)$ time, resulting in total complexity

$$\mathcal{O}\left(\frac{mn}{\log_2 n}\right).$$

An explicit and convenient upper bound on the total complexity is $\dfrac{4mn}{\log_2 n}$ [48].