

Better Binary Tree Codes From MDS Tree Codes

Tal Yankovitz*

Abstract

Finding good tree code constructions is among the most natural questions in error-correcting codes over the Hamming channel. Unlike the usual setting, where one encodes a single message against adversarial errors, tree codes allow a sender to encode and send a *stream*, in an *online* manner, while protecting it from adversarial errors. Moreover, Schulman (STOC 93) showed that tree codes yield schemes which protect any two-party communication protocol from adversarial errors. Schulman asked whether there are *explicit* tree code families with constant distance and constant rate. While this question remains open, in a breakthrough, Cohen, Haeupler and Schulman (STOC 2018) constructed explicit constant-distance tree codes with rate $1/O(\log \log n)$, where n denotes the message length, improving upon the $1/O(\log n)$ previously known. Subsequently, Ben-Yaacov, Cohen and Yankovitz (STOC 22) improved the rate-distance tradeoff, constructing explicit distance- δ tree codes with rate $1/O(\sqrt{\delta} \log \log n)$.

In this work we construct explicit distance- δ tree codes with rate $1/O(\delta \log \log n)$. In particular, this implies explicit tree codes with constant rate and distance $1/O(\log \log n)$, improving upon the previously known $1/O(\log^2 \log n)$. To achieve this improved rate-distance tradeoff, new ideas are required. Specifically, we devise a construction whose key idea relies on *MDS tree codes*. MDS tree codes were first considered by Pudlák (Linear Algebra Appl. 16), who gave an $\mathbb{F}_2$ -linear MDS tree code construction with alphabet size $2^{O(n^3)}$. As another contribution, we present such a construction with alphabet size $2^{O(n^2)}$.

*University of Toronto. tal.yankovitz@utoronto.ca. The research leading to the results in this paper was done while the author was at UT Austin and supported by NSF Grant CCF-2312573.

Contents

1	Introduction	1
1.1	Our main contribution	2
1.1.1	Remarks	3
1.2	Our second contribution - large-alphabet MDS tree codes	3
1.2.1	MDS tree codes	3
1.3	Prior work	5
1.3.1	[CHS18]	5
1.3.2	[BYCY22]	5
1.4	Technical overview: improved tree codes	6
1.4.1	The “multi-scale tree code”	6
1.4.2	From the multi-scale tree code to the final construction	9
1.5	Technical overview: $\mathbb{F}_2$ -linear MDS tree codes over a smaller alphabet	10
1.6	A remark on the applicability of the rate-immediacy [CSS25] barrier	11
2	Tree codes with better rate	11
2.1	Preliminaries	11
2.2	The construction	11
2.2.1	A multi-scale MDS tree code	13
2.2.2	A ρ -schedule	15
2.2.3	The main construction	16
3	MDS tree codes over a smaller alphabet	23
A	Auxiliary proofs	28

1 Introduction

Constructing tree codes is an important problem in error correcting codes. A vast effort has been devoted to the classical question of error correcting codes for encoding *messages*, in order to allow their decoding amid adversarial, or worst-case, errors. A tree code, on the other hand, deals with a scenario where one needs to encode a *stream*, in an *online* fashion. That is, the encoded message is not known in full ahead of time; rather, it is revealed to the speaking party bit by bit. On the other, receiving, end, the requirement is to receive each such bit in a timely manner, promptly after it is known to the encoder, and before the next bit is revealed. For example, one can think of a live radio program, transmitted over a broadcast channel, to a multitude of listeners.

The aforementioned scenario corresponds to a one-way communication setting, whereas the utility of tree codes extends beyond that, to another elementary problem in error correcting codes: that of *protocol-wise error resilience*, in contrast to *message-wise error resilience*. In fact, this was the original context in which tree codes were introduced, in the seminal work of Schulman [Sch93].

We define tree codes through their encoding procedure, used for processing an input $x \in \{0, 1\}^n$ of length n . Such an encoding procedure is *online*, and maps each x to a tree-code codeword $\mathcal{TC}(x) \in \Sigma^n$. An online encoding cannot guarantee that every two codewords differ on, say, a constant fraction of locations; indeed, one can consider two messages that differ only on the n -th bit. What the encoding procedure is required to satisfy is, in a sense, the strongest *possible* guarantee. That is, for every two distinct inputs x and x' , the codewords $\mathcal{TC}(x)$ and $\mathcal{TC}(x')$ differ significantly, on a δ -fraction of locations, on every *interval which starts at the first divergence of x, x' , and ends some time after it*. We state the formal definition.

Definition 1.1 (binary tree code with distance δ and rate ρ). *A tree code of length n and (relative) distance δ is an online function*

$$\mathcal{TC} : \{0, 1\}^n \rightarrow (\{0, 1\}^r)^n$$

such that the following holds. For every distinct $x = (x_1, \dots, x_n), x' = (x'_1, \dots, x'_n) \in \{0, 1\}^n$, let

$$s = \min\{i \in [n] : x_i \neq x'_i\}.$$

Then for every $h \in [n - s + 1]$,

$$\text{HammingDist}(\mathcal{TC}(x)_{s, \dots, s+h-1}, \mathcal{TC}(x')_{s, \dots, s+h-1}) \geq \delta h.$$

By online we mean that for every $t \in [n]$, the value $\mathcal{TC}(x)_t$, denoted $\mathcal{TC}^t(x_1, \dots, x_t)$, depends only on $x_1, \dots, x_t$. We call

$$\rho := \frac{1}{r}$$

the rate of the tree code. The index s is called the split of x and x' .

As is standard, we are mainly interested in a *family* of such encodings, for infinitely many values of n , and we will often use “tree code” as short for “tree code family”. As mentioned, tree codes were first defined by Schulman [Sch93], who showed that they yield interactive schemes for achieving error-resilient two-party protocols. A tree code family is called *good* if its rate and (relative) distance are constants bounded away from 0. A priori, it is not clear that good tree codes ever *exist*. Schulman [Sch93, Sch96] showed that they do, and asked: **are there *explicit* good tree codes?**¹

¹Here, by explicit we mean that given $x \in \{0, 1\}^n$ the codeword $\mathcal{TC}(x)$ can be computed in deterministic time $\text{poly}(n)$.

The question has been open since.

Motivated by Schulman’s application, the problem of finding tree code constructions has attracted significant attention [Sch94, Bra12, MS14, Pud16, GHK⁺16, CHS18, NW20, BH20, BYCN21, Pud22, BYCY22, MRR25, CSS25]. Although explicit good tree codes have not been discovered, significant progress has been achieved by the aforementioned works. In their surprising breakthrough, Cohen, Haeupler and Schulman showed the following.

Theorem 1.2 ([CHS18]). *For every $\delta < 0.999$,² there is an explicit tree code family with distance δ and rate*

$$\rho = \rho(n) \geq \frac{1}{c \log \log n},$$

where $c > 1$ is a universal constant.

This constituted an exponential improvement over the previously known rate [Sch94], namely $\rho = \Omega(1/\log n)$.

Subsequently, Cohen, Ben-Yaacov and Yankovitz [BYCY22] obtained an improved state-of-the-art construction.

Theorem 1.3 ([BYCY22]). *For every $\delta < 0.999$, there is an explicit tree code family with distance δ and rate*

$$\rho = \rho(n) \geq \frac{1}{c\sqrt{\delta} \log \log n},$$

where $c > 1$ is a universal constant.

In particular, their result yielded an explicit tree code family with rate $\rho = \Omega(1)$ and distance

$$\delta = \frac{1}{\Omega(\log^2 \log n)}.$$

This was a significant improvement over the prior state-of-the-art distance for constant-rate explicit tree codes, which was $\delta = 1/\Omega(\log n)$, by the work of Gelles, Haeupler, Kol and Ron-Zewi [GHK⁺16].

1.1 Our main contribution

Our main result is the following.

Theorem 1.4. *For every $\delta < 0.999$, there is an explicit tree code family with distance δ and rate*

$$\rho = \rho(n) \geq \frac{1}{c\delta \log \log n},$$

where $c > 1$ is a universal constant.

In particular, there is an explicit tree code family with rate $\Omega(1)$ and distance

$$\delta = \Omega\left(\frac{1}{\log \log n}\right).$$

Thus, our result yields an improved state of the art in both regimes: the constant-distance regime and the constant-rate regime.

²0.999 is arbitrary.

1.1.1 Remarks

Finite vs infinite tree codes. Here, we have defined tree codes as having finite length n . In an earlier, stronger definition, tree codes encode an infinite stream. However, it was shown in [BYCY22] that an efficient construction of finite-length tree codes can be transformed into an infinite construction with almost the same parameters. We refer to [BYCY22] for the details.

Tree codes as an encoding vs a coloring of a binary tree. Here, we have defined tree codes by the message encoding function. In a more classical definition, in the infinite case, they are presented as an edge-coloring of the infinite rooted complete binary tree using colors from a prescribed set of colors; in the finite case, one colors a rooted binary tree of finite length. Under this definition, one considers any two same-depth vertices of the tree, and requires that the two paths from their lowest common ancestor are colored differently on at least a δ -fraction of the edges.

The two definitions, tree codes via an encoding function and tree codes as colorings of a binary tree, are readily verified to be equivalent, as an online encoding function induces a coloring of the tree, and vice versa. In the tree-coloring language, what we called the “length” of a finite tree code is referred to as its “depth”, and the output alphabet size 2^r , where $\rho = 1/r$ is the rate, is the “number of colors”. Thus, a good tree code induces a coloring of infinitely many binary trees using only finitely many colors, while satisfying the aforementioned guarantee with $\delta = \Omega(1)$.

From this more graph-theoretic viewpoint, the result of [CHS18] gave the first explicit coloring, admitting constant δ , which uses a number of colors polylogarithmic in the tree depth n . The result of [BYCY22] achieved a sub-logarithmic construction using $\log^{O(\sqrt{\delta})} n$ colors; our result yields an explicit coloring with $\log^{O(\delta)} n$ colors.

1.2 Our second contribution - large-alphabet MDS tree codes

To state our second contribution, we first extend the definition of tree codes to non-binary alphabets.

Definition 1.5. *A (non-binary) tree code of length n and (relative) distance δ over an alphabet Γ is an online function*

$$\mathcal{TC} : \Gamma^n \rightarrow (\Gamma^r)^n$$

such that the following holds. For every distinct $x = (x_1, \dots, x_n), x' = (x'_1, \dots, x'_n) \in \Gamma^n$, let

$$s = \min\{i \in [n] : x_i \neq x'_i\}.$$

Then for every $h \in [n - s + 1]$,

$$\text{HammingDist}(\mathcal{TC}(x)_{s, \dots, s+h-1}, \mathcal{TC}(x')_{s, \dots, s+h-1}) \geq \delta h.$$

We call $\rho := 1/r$ the rate of the tree code.

1.2.1 MDS tree codes

In the study of error correcting codes, MDS codes have been an area driving much of the obtained progress, with Reed-Solomon codes being the most prominent and well-studied example. Since much is unknown about tree codes, asking questions about MDS tree codes, which are in a sense perfect, seems like a natural avenue.

In his important work, Pudlák [Pud16] initiated the study of linear MDS tree codes. He proved a lower bound showing that, over any input alphabet, a tree code of length n and rate $1/2$ can have distance at most $1/2 + 1/(2n)$.

Pudlák asked: **are there families of tree codes achieving the same rate and distance, most desirably while being over a small alphabet Γ ?** We stress that this question concerns the information-theoretic level, even prior to efficiency considerations.

Pudlák showed that the answer is *affirmative*, for finite but not constant alphabets Γ .

Theorem 1.6 ([Pud16]). *For every $n, \ell \in \mathbb{N}$ such that $\ell > n(n-1)^2 = \Theta(n^3)$ there exists an explicit linear tree code $\mathcal{TC} : (\{0, 1\}^\ell)^n \rightarrow (\{0, 1\}^\ell \times \{0, 1\}^\ell)^n$ with distance $\frac{1}{2} + \frac{1}{2n}$.*

That is, he showed that there are infinitely many MDS tree codes, with alphabet size $2^{\Theta(n^3)}$. Here and throughout, linearity means $\mathbb{F}_2$ -linearity.

Pudlák's construction, which is explicit and linear, is based on algebraic idea that requires the underlying finite field to be large enough. We remark that constructions which are MDS over an infinite alphabet are also known [Pud16, CHS18, NW19]. Moreover, [CHS18] show how to transform a specific such construction which is over $\mathbb{Z}$ into a tree code over an alphabet of size 2^n (where n is the length of the tree code); however, the resulting tree code is no longer MDS.

Given the importance of finding constructions of MDS tree codes with smaller alphabets, our second contribution is a construction of infinitely many MDS tree codes with alphabets of size at most 2^{n^2} , improving upon [Pud16], with one caveat. The caveat is that our construction is variable-alphabet: the i -th input symbol may come from its own alphabet Γ_i . However, every Γ_i satisfies the stated bound.

Definition 1.7. *A (variable-alphabet) tree code of length n and (relative) distance δ , over alphabets $\Gamma_1, \dots, \Gamma_n$, is an online function*

$$\mathcal{TC} : \Gamma_1 \times \dots \times \Gamma_n \rightarrow \Gamma_1^r \times \dots \times \Gamma_n^r$$

such that the following holds. For every distinct $x = (x_1, \dots, x_n), x' = (x'_1, \dots, x'_n) \in \Gamma_1 \times \dots \times \Gamma_n$, let

$$s = \min\{i \in [n] : x_i \neq x'_i\}.$$

Then for every $h \in [n - s + 1]$,

$$\text{HammingDist}(\mathcal{TC}(x)_{s, \dots, s+h-1}, \mathcal{TC}(x')_{s, \dots, s+h-1}) \geq \delta h.$$

We call $\rho := 1/r$ the rate of the tree code.

Our second contribution is as follows.

Theorem 1.8 (See Theorem 3.1). *For every $n \in \mathbb{N}_{\geq 3}$, there exist alphabets*

$$\Gamma_i = \{0, 1\}^{c_i}, \quad i \in [n],$$

with $|\Gamma_i| < 2^{n^2}$ for every $i \in [n]$, and an explicit linear (variable-alphabet) tree code

$$\mathcal{TC}_n : \Gamma_1 \times \dots \times \Gamma_n \rightarrow (\Gamma_1 \times \Gamma_1) \times \dots \times (\Gamma_n \times \Gamma_n)$$

with rate $1/2$ and distance $1/2 + 1/(2n)$.³

Beyond being MDS and $\mathbb{F}_2$ -linear, the construction is of an elementary nature, which we view as an advantage: it consists only of shifting binary vectors and taking their sum. More generally, since further reducing the alphabet size of explicit tree codes is an important direction, simple ideas may be advantageous in pursuing this goal.

³An earlier version of the construction used to prove Theorem 1.8 appeared in [Yan25], which is subsumed by this report.

1.3 Prior work

Before giving a technical overview of our contribution, we give technical details on prior work.

1.3.1 [CHS18]

The result of [CHS18] is achieved through a two-step construction. The first step is to construct a *large-alphabet* tree code, and the second step is to incorporate it within an *alphabet-reduction* framework. A large-alphabet tree code has input alphabet $\{0, 1\}^\ell$ instead of $\{0, 1\}$, as in Definition 1.1, where ℓ may be a function of the length n .

First, to obtain their large-alphabet tree code, [CHS18] encode a given message as a polynomial $p \in \mathbb{R}[x]$ over the Newton basis, which is $\left\{\binom{x}{i}\right\}_{i \in \mathbb{N} \cup \{0\}}$, rather than the standard basis $\{x^i\}_{i \in \mathbb{N} \cup \{0\}}$, usually used in polynomial-based error correcting codes. To bound the distance of these tree codes, [CHS18] show that, for polynomials expressed in the Newton basis *over the reals*, the Gessel-Viennot Lemma can be employed to bound the number of integral roots in terms of the number of nonzero coefficients, i.e., the sparsity. This establishes that their large-alphabet tree code has constant distance.

Second, we describe the alphabet-reduction step of [CHS18]. The main concept used in this step is their notion of a *lagged* tree code. A lagged tree code, with lag L , is required to satisfy the distance guarantee only for intervals of length at least L . They observe that the large-alphabet tree code, concatenated with a standard error correcting code, can be used to construct a tree code of length n and lag $O(\sqrt{n})$. Their final tree code then writes down, in parallel, the outputs of $O(\log \log n)$ such lagged tree codes, of decreasing lengths.

1.3.2 [BYCY22]

To achieve their result, [BYCY22] take a more refined approach compared to the alphabet-reduction framework of [CHS18]. While still relying on the Newton basis as a large-alphabet tree code, as in [CHS18], they observe that, under a careful analysis, its rate can be viewed as a parametrized function depending on the alphabet size and the tree code length, where initially the dependence on the length is unfavorable. They then seek to improve this length dependence. Towards that, they devise a transformation which takes a tree code admitting such a parametrization, and yields a tree code whose rate's dependence on the length is improved. Repeating their transformation $O(\log \log n)$ times produces their result.

The aforementioned length-dependence reducing transformation of [BYCY22] is reminiscent of code tensoring, though it is significantly more involved. It consists of several encoding steps. It works by writing down the input to a given tree code as the rows of a matrix. To each row, one applies the tree code which is given as input. To encode the columns, one uses a large-alphabet tree code, where every row is viewed as a single large symbol. These two parts of the encoding do not suffice by themselves, and there are further nuances which are handled in [BYCY22]. In particular, one must work with *systematic* tree codes, as otherwise the steps above would already blow up the rate if performed $O(\log \log n)$ times.

One issue which arises in the transformation is the case of a split near the end of a row. In this case, the tree code applied to each row cannot guarantee distance soon after the split, at points in time after the row ends. To overcome this issue, [BYCY22] incorporate several ideas which work together. One of these is that when encoding every row, one also outputs a Reed–Solomon-like encoding of its preceding row, while making sure that if the split was close to the end of the preceding row, the Reed–Solomon code has higher distance, to make up for that.

1.4 Technical overview: improved tree codes

The challenge. Both [CHS18] and [BYCY22] work by outputting, at each point in time, symbols which are the result of $O(\log \log n)$ tree codes simulated in parallel. At a high level, [BYCY22] achieve a saving, and are able to have only $O(\sqrt{\delta} \log \log n)$ bits of redundancy, by having the different-level tree codes share a common systematic part, and by maintaining that every redundancy symbol corresponding to one of the levels is of size an $O(\sqrt{\delta})$ -fraction of the input-symbol length. However, the framework of [BYCY22] inherently cannot achieve redundancy $O(\delta \log \log n)$, even by attempting to further optimize its parameters.

The inherent reason is that, in every step there, at least part of the redundancy comes from applying a large-alphabet tree code and concatenating its output with an error correcting code. This concatenation is crucial because the distance needs to hold at the symbol level. However, as is the case with concatenation schemes, if the large-alphabet tree code has distance δ_1 and the error correcting code has distance δ_2 , then, at best, the concatenated code has rate approximately

$$(1 - \delta_1)(1 - \delta_2)$$

and distance $\delta_1 \delta_2$. Here, rate is the ratio between the input length of the code k , to its output length n . It follows that code rate $(1 - \delta_1)(1 - \delta_2)$ corresponds to *redundancy rate*

$$\frac{n - k}{n} = \frac{1}{(1 - \delta_1)(1 - \delta_2)} - 1 = \frac{\delta_1 + \delta_2 - \delta_1 \delta_2}{(1 - \delta_1)(1 - \delta_2)} \geq \delta_1 + \delta_2 - \delta_1 \delta_2 \geq \frac{1}{2}(\delta_1 + \delta_2),$$

since $\delta_1, \delta_2 \in [0, 1]$. Thus, to achieve redundancy rate at most α , it must be that $\delta_1, \delta_2 \leq 2\alpha$, and therefore the distance $\delta_1 \delta_2$ is at most $4\alpha^2$. This explains why, to beat the $O(\sqrt{\delta} \log \log n)$ rate, new ideas are needed.

Moreover, this barrier applies to a core element of the [CHS18] framework itself, namely that at each level a large-alphabet tree code is concatenated with an error correcting code, and not only to the later framework of [BYCY22]. Thus, the fact that a higher rate can be achieved was not at all clear a priori.

1.4.1 The “multi-scale tree code”

As mentioned, previous constructions wrote down, *in parallel*, $L = O(\log \log n)$ copies of different tree codes. The idea in this approach is that, no matter how far one is from the split s , one of the levels is effective. That is, the tree codes of the different levels have increasing block sizes

$$\ell_1 < \dots < \ell_L.$$

The high-level idea, sweeping certain key points under the rug, is that given an interval of length h ,

$$I = [s, s + h - 1] := \{s, \dots, s + h - 1\},$$

if one considers the highest level $a \in [L]$ such that $\ell_a \leq h$, then the a -th level tree code assures distance in the I -part of the output, $\mathcal{TC}(x)_I$.

Our main idea is the following: instead of separately writing L outputs corresponding to the different tree codes, we obtain a *single* construction which provides, *simultaneously*, a distance guarantee for all L levels, while being as economical in rate as possible. We do so by using linear MDS tree codes and a recursive composition trick. We now explain the central ideas of this construction.

A linear MDS tree code. We represent a linear MDS tree code with block length ℓ_a by an online linear map

$$\mathcal{R} : \mathbb{F}_{2^{\ell_a}}^n \rightarrow \mathbb{F}_{2^{\ell_a}}^n,$$

where the map R_t corresponding to the t -th output of $\mathcal{R}$, namely $\mathcal{R}(x)_t = R_t(x_{\leq t})$, depends only on $x_{\leq t}$. The induced tree code encodes $x = (x_1, \dots, x_n)$ by outputting, at each time t ,

$$(x_t, R_t(x_{\leq t})).$$

The MDS guarantee is that for every nonzero $x = (x_1, \dots, x_n) \in (\mathbb{F}_{2^{\ell_a}})^n$, if $i = \min\{t : x_t \neq 0\}$, then for every suffix length h ,

$$|\{t \in [i, i+h-1] : x_t \neq 0\}| + |\{t \in [i, i+h-1] : R_t(x_{\leq t}) \neq 0\}| > h. \quad (1.1)$$

The induced tree code has rate $1/2$ (recalling that the encoding is linear). By arguments given in [Pud16], such a representation is without loss of generality.

A “normal” tree code. We will call a linear MDS tree code, given by a linear map

$$\mathcal{R} : \mathbb{F}_{2^{\ell_a}}^n \rightarrow \mathbb{F}_{2^{\ell_a}}^n,$$

normal if, writing

$$\mathcal{R}(x)_t = R_t(x_1, \dots, x_t),$$

for a linear map

$$R_t : \mathbb{F}_{2^{\ell_a}}^t \rightarrow \mathbb{F}_{2^{\ell_a}},$$

we have, for every $t \in [n]$,

$$R_t(x_1, \dots, x_t) = x_t + \sum_{j=1}^{t-1} \gamma_{t,j} x_j$$

for some $\gamma_{t,j} \in \mathbb{F}_{2^{\ell_a}}$, $j \in [t-1]$. That is, the way R_t depends on the current, t -th input is by adding it with coefficient 1. We show, by a simple linear-algebra argument, that a linear MDS tree code can be made normal, so assuming that one is at hand is without loss of generality.

The multi-scale tree code. For any $\ell \in \mathbb{N}$, we will implicitly interchange, whenever needed, between elements of $\mathbb{F}_2^\ell$ and of $\mathbb{F}_{2^\ell}$, by fixing some $\mathbb{F}_2$ -linear isomorphism between them. We describe the multi-scale construction idea for two levels; the full construction follows inductively.

Assume that an MDS linear normal tree code over alphabet $\mathbb{F}_{2^{\ell_a}}$ and length $n_a = n/\ell_a$, is given by

$$y \mapsto (y, \mathcal{R}(y)), \quad \mathcal{R} : \mathbb{F}_{2^{\ell_a}}^{n_a} \rightarrow \mathbb{F}_{2^{\ell_a}}^{n_a}.$$

Assume further that a second MDS linear normal tree code over the smaller alphabet $\mathbb{F}_{2^{\ell_{a-1}}}$ and length $n_{a-1} = \ell_a/\ell_{a-1}$ is given by

$$z \mapsto (z, \mathcal{R}'(z)), \quad \mathcal{R}' : \mathbb{F}_{2^{\ell_{a-1}}}^{n_{a-1}} \rightarrow \mathbb{F}_{2^{\ell_{a-1}}}^{n_{a-1}}.$$

We define the induced two-level encoding. For an input $x = (x_1, \dots, x_n) \in \mathbb{F}_2^n$, decompose $[n]$ into consecutive blocks of size ℓ_a ,

$$[n] = Q_1 \cup \dots \cup Q_{n/\ell_a},$$

and further decompose each ℓ_a -block into consecutive subblocks of size ℓ_{a-1} ,

$$[\ell_a] = Q'_1 \cup \dots \cup Q'_{\ell_a/\ell_{a-1}}.$$

The encoding is

$$x \mapsto \left(x, \mathcal{R} \left(\mathcal{R}'((x_{Q_1})_{Q'_1}, \dots, (x_{Q_1})_{Q'_{\ell_a/\ell_{a-1}}}), \right. \right. \\ \dots, \\ \left. \left. \mathcal{R}'((x_{Q_{n/\ell_a}})_{Q'_1}, \dots, (x_{Q_{n/\ell_a}})_{Q'_{\ell_a/\ell_{a-1}}}) \right) \right).$$

That is, instead of having the redundancy map of the higher-level tree code act on the blocks of the input, $x_{Q_1}, \dots, x_{Q_{n/\ell_a}}$, it acts on the result of applying the smaller-level redundancy map to each block. Observe that this composition does not affect the rate, which remains $1/2$.

Why does the above work? The question is well placed, as one may observe that nowhere in the described encoding is a complete output of the higher-level tree code embedded. Indeed, the systematic part is the same as that of the given higher-level tree code, while the redundancy map has been altered. However, the correctness of the construction stems from the properties of normal MDS tree codes. As usual, since the construction is linear, to show that a distance guarantee holds, it suffices to bound the number of nonzero outputs given a nonzero input, where the split is the index of its first nonzero element. We make two points.

- We argue that an output of the smaller-level tree code *is* embedded in the output, though not at every time, but rather at the one time at which it is needed. The output of the smaller-level tree code is only needed in order to provide distance at times which are close to the split s . For intervals starting from s and having length $h > \ell_a$, the larger-level tree code is in effect. However, since we assumed that the larger-level tree code is normal, the way its t -th redundancy, R_t , depends on the current input is by addition with coefficient 1. At the time of the split, all earlier inputs are 0. Therefore, at the time of the split s , R_s is in fact the identity map on x_s . Since in our construction we set the smaller-level tree code redundancy map as the input to the larger-level tree code redundancy map, it follows that a valid copy of the smaller-level redundancy map is output at the time of the split. The same is not true for later times.
- While the smaller-level tree code is not needed after its first use following the split, this does not a priori mean that applying the larger-level redundancy map to it, rather than to the input itself, does not hurt the larger-level tree code guarantee, which is needed for those later times. However, we argue that the larger-level tree code distance guarantee does in fact manifest in the two-level construction, and more specifically, the MDS guarantee [Equation \(1.1\)](#).

The reason is that the redundancy map $\mathcal{R}'$ of the smaller-level tree code must be invertible, an observation following from it being MDS. Moreover, the number of nonzero redundancy outputs assured by [Equation \(1.1\)](#) depends only on the number of nonzero elements among $x_1, \dots, x_t$: the fewer nonzero elements there are among them, the more nonzero redundancy outputs there must be. Applying an invertible linear map to each element of $(x_1, \dots, x_t)$ does not change its pattern of nonzero entries. Therefore, applying the larger-level tree code redundancy map to the output produced by the smaller-level tree code, rather than to the input itself, maintains the same guarantee as in [Equation \(1.1\)](#).

The two-level construction above, and the accompanying reasoning, can be extended inductively to the case of L levels, by setting appropriate scales

$$\ell_1 \mid \cdots \mid \ell_L \mid n.$$

The exact details and argument appear in the technical section.

1.4.2 From the multi-scale tree code to the final construction

The multi-scale tree code construction assures distance, after the time of the split, on growing scales. We view it as an interesting question whether this combinatorial object may have direct applications. To use it in our construction, we need to translate the growing-scale distance guarantee, which concerns the number of nonzero blocks, to binary distance. Here, a block is considered nonzero if it is nonzero somewhere. This will require applying an error correcting code. However, since the multi-scale construction is a *single* construction for all levels together, we evade the composition barrier described above.

Despite ending up with a more favorable rate tradeoff, we still have to pay proportionally to the number of levels when applying these error correcting codes. As in [CHS18, BYCY22], an error correcting code applied to an output block can only be applied after that block has been computed. However, as in [BYCY22], the way in which we output each such error correcting code following the block completion must be chosen carefully, to avoid having points in time where all levels produce such an output.

The work of [BYCY22] solves this issue by working through alphabets of moderate size as a precursor to achieving a binary alphabet, and by employing an idea of “spreading” the extra information that is output in a balanced, averaged manner. We take a slightly different approach here, which we view as clean within the context of our construction: using a **schedule**. The schedule decides, at each point in time, which of the preceding blocks “get to” output a bit of their error correcting code information. It turns out that there is a way to compute such a schedule which assures that, at any point in time, at most $O(\delta L)$ preceding blocks “speak”, while satisfying the guarantee that for every block, before its succeeding block terminates, all of its error correcting code information has been output.

One more point we need to handle is that, by relying on distance guarantees that arise from different scales, smaller tree codes have not only smaller block length, but also shorter length. This is a problem in cases where the split is close to such a tree code’s end, as its distance guarantee stops holding once this end is reached. In fact, this is a problem handled in the construction of [CHS18], and a similar issue arises in [BYCY22], which solves it using a different mechanism.

We overcome this issue with an approach more similar to [CHS18], with an important change. The idea of [CHS18] is, at each level, to have two instances of the encoding, where the blocks of one instance are shifted compared to the blocks of the other. This assures that one of the two instances is always far away from a tree code’s end. If we were to employ such a shift at every level, it would be too costly in information bits. We show that, in fact, it is possible to have only *two* shifted instances for all levels altogether: for every one of the L scales, at any point in time, at least one of the two instances is far away from the next end.

We mention that there may be other solutions to this issue, such as an approach similar to the more involved one taken by [BYCY22], which is tailored to their needs.

1.5 Technical overview: $\mathbb{F}_2$ -linear MDS tree codes over a smaller alphabet

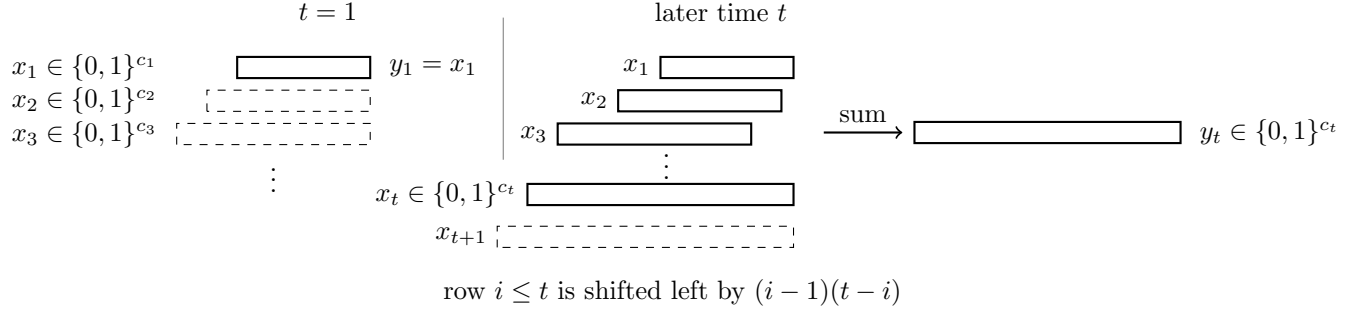


Figure 1: *Visualization of the shift-and-sum operation. At time t , only shifts of rows $x_1, \dots, x_t$ participate; future rows are dashed. The result y_t is the column-wise binary sum, where empty entries are considered to be 0. The length c_t is chosen so that $y_t \in \{0, 1\}^{c_t}$.*

We now give a brief overview of the construction behind [Theorem 1.8](#). The input is

$$x = (x_1, \dots, x_n),$$

where $x_t \in \Gamma_t = \{0, 1\}^{c_t}$ is a binary vector. The encoding in our construction is systematic. That is, at time t we output

$$(x_t, y_t),$$

where $y_t \in \Gamma_t$ is a redundancy symbol.

Fix a time $t \in [n]$. The output y_t is computed in the following manner, where we will only use the symbols $x_1, \dots, x_t$, which we think of as positioned in a matrix-like structure (see [Figure 1](#)). Specifically, each vector x_i , for $i \leq t$, is placed in a row of length c_t , initially aligned to the right. We then shift this row

$$(i-1)(t-i)$$

places to the left, filling the blank coordinates with zeroes. We set the output vector y_t to be the $\mathbb{F}_2$ -sum of these shifted rows.

In the technical part, the lengths $c_1, \dots, c_n$ are chosen so that the number of columns at time t , after the row shifts, which corresponds to the length of y_t , is exactly equal to c_t , and additionally so that $c_t < n^2$, yielding the claimed alphabet sizes.

As the construction is evidently (bitwise) linear, it only remains to explain why this satisfies the MDS tree code distance guarantee. Fix a nonzero input x and let

$$s = \min\{i \in [n] \mid x_i \neq \bar{0}\}$$

be its split. For the systematic encoding $x_t \mapsto (x_t, y_t)$, the MDS guarantee amounts to showing that, on every interval beginning at s , the number of zero redundancy symbols is strictly smaller than the number of nonzero input symbols.

We prove this by induction on the number of nonzero input symbols among $x_1, \dots, x_n$. The crux of the induction step is to understand what happens at a time t for which the redundancy symbol vanishes, namely $y_t = \bar{0}$. Look at the matrix formed by the shifted copies of $x_1, \dots, x_t$ at this time. Among all columns which contain a nonzero entry, take the left-most one. Let i be the

smallest row index which is nonzero in this column. Since the column sum is zero, there must be another row $j > i$ which is also nonzero in the same column.

Now compare the future shifts of rows i and j . When time increases, row r gains further leftward shift at speed $r - 1$. Since $j > i$, row j moves left faster than row i . Therefore, after time t , row i can no longer participate in canceling the left-most nonzero shifted column, as row j is further to its left. So, as x_i is “out of the game”, there is one less nonzero row at play, and the induction hypothesis can be employed. A formal account appears in [Section 3](#).

1.6 A remark on the applicability of the rate-immediacy [\[CSS25\]](#) barrier

Recently, [\[CSS25\]](#) proved a lower bound relating the achievable rate of tree codes to what they call the “rate-immediacy” property. This bound applies, in particular, to tree codes obtained through the frameworks of [\[CHS18\]](#) and [\[BYCY22\]](#). Similarly, our improved-rate binary construction has a code-composition part, and therefore also satisfies the relevant immediacy property. The work of [\[CSS25\]](#) establishes that, to improve the $1/\log \log n$ factor in the rate, such an immediacy property needs to be surpassed. Indeed, our construction features the fact that the composed error-correcting-code bits constitute the *only* significant blow-up.

We add that the rate-immediacy barrier is not applicable to the multi-scale (non-binary distance) tree code construction or to our large-alphabet MDS tree code construction.

2 Tree codes with better rate

2.1 Preliminaries

We write $\mathbb{N} = \{1, 2, \dots\}$ for the set of natural numbers. All logarithms are taken to base 2. For $n \in \mathbb{N}$, we write $[n] := \{1, \dots, n\}$. For integers $i \leq j$, we write $[i, j] := \{i, i + 1, \dots, j\}$. For two vectors or strings $u, v \in \Sigma^n$, we use $\text{Dist}(u, v)$ to denote their absolute Hamming distance, and $u \circ v$ to denote their concatenation. Given $u \in \Sigma^n$ and $i \leq j$, we use $u_{i, \dots, j}$ to denote the substring $(u_i, \dots, u_j)$. We write $\bar{0}$ for the all-zero vector, with its length clear from context. For a binary string z , we write $\text{HammingWeight}(z)$ for the number of nonzero entries of z .

2.2 The construction

We now turn to the construction of tree codes with improved rate. For simplicity, we first prove [Theorem 2.8](#), which gives a slightly weaker binary construction, with rate $\Omega(1/(H(\delta) \log \log n))$ instead of $\Omega(1/(\delta \log \log n))$. In [Remark 2.10](#), we explain how to extend this to the stated parameters.

We start by defining MDS tree codes, which were considered by [\[Pud16\]](#). The definition below is specialized to the case of rate-1/2 linear MDS tree codes; throughout this section, we will simply write “MDS tree code” to mean this case.

Definition 2.1 (MDS tree code). *Let $n, \ell \in \mathbb{N}$. We say that a systematic online linear map*

$$T : (\{0, 1\}^\ell)^n \longrightarrow (\{0, 1\}^\ell \times \{0, 1\}^\ell)^n$$

is an MDS tree code if, writing

$$T(x)_t = (x_t, R_t(x_{\leq t})),$$

for linear maps

$$R_t : (\{0, 1\}^\ell)^t \rightarrow \{0, 1\}^\ell, \quad t \in [n],$$

the following holds. For every nonzero $x = (x_1, \dots, x_n) \in (\{0, 1\}^\ell)^n$, let

$$i = \min\{t \in [n] \mid x_t \neq \bar{0}\}.$$

Then for every $h \in [n - i + 1]$,

$$|\{j \in [i, i + h - 1] : x_j \neq \bar{0}\}| + |\{j \in [i, i + h - 1] : R_j(x_{\leq j}) \neq \bar{0}\}| > h. \quad (2.1)$$

The construction of Pudlák [Pud16] implies the following. For completeness, we give a proof sketch following [Pud16] for the binary case in Appendix A.

Theorem 2.2 ([Pud16]). *For every $n, \ell \in \mathbb{N}$ such that $\ell \geq n^3$, there exists an explicit linear MDS tree code*

$$T : (\{0, 1\}^\ell)^n \rightarrow (\{0, 1\}^\ell \times \{0, 1\}^\ell)^n.$$

We call an MDS tree code *normal* if, writing

$$T(x)_t = (x_t, R_t(x_{\leq t})),$$

we have, for every $t \in [n]$,

$$R_t(x_{\leq t}) = x_t + F_t(x_{<t})$$

for some $\mathbb{F}_2$ -linear map F_t . Any linear systematic MDS tree code can be made normal.

Claim 2.3. *For every $n \in \mathbb{N}$, there exists an explicit normal linear MDS tree code*

$$T : (\{0, 1\}^\ell)^n \rightarrow (\{0, 1\}^\ell \times \{0, 1\}^\ell)^n$$

with $\ell = n^3$. That is, writing

$$T(x)_t = (x_t, R_t(x_{\leq t})),$$

we have, for every $t \in [n]$,

$$R_t(x_{\leq t}) = x_t + F_t(x_{<t})$$

for some $\mathbb{F}_2$ -linear map F_t .

Proof. Start with the code from Theorem 2.2, and write

$$T(x)_t = (x_t, \tilde{R}_t(x_{\leq t})).$$

Since $\tilde{R}_t : (\{0, 1\}^\ell)^t \rightarrow \{0, 1\}^\ell$ is linear, we can write

$$\tilde{R}_t(x_{\leq t}) = A_t(x_t) + G_t(x_{<t}),$$

where

$$A_t : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell, \quad G_t : (\{0, 1\}^\ell)^{t-1} \rightarrow \{0, 1\}^\ell$$

are $\mathbb{F}_2$ -linear maps.

The map A_t is invertible. Indeed, if $A_t(u) = \bar{0}$ for some $u \neq \bar{0}$, then the input with $x_t = u$ and $x_j = \bar{0}$ for all $j \neq t$ contradicts the MDS condition on the length-one interval $[t, t]$.

Define a new redundancy coordinate by

$$R_t(x_{\leq t}) := A_t^{-1}(\tilde{R}_t(x_{\leq t})) = x_t + A_t^{-1}(G_t(x_{<t})).$$

Thus $R_t(x_{\leq t}) = x_t + F_t(x_{<t})$, for the linear map $F_t = A_t^{-1} \circ G_t$. Since A_t^{-1} is invertible, replacing $\tilde{R}_t$ by R_t does not change which redundancy symbols are zero. Therefore the MDS property is preserved. $\square$

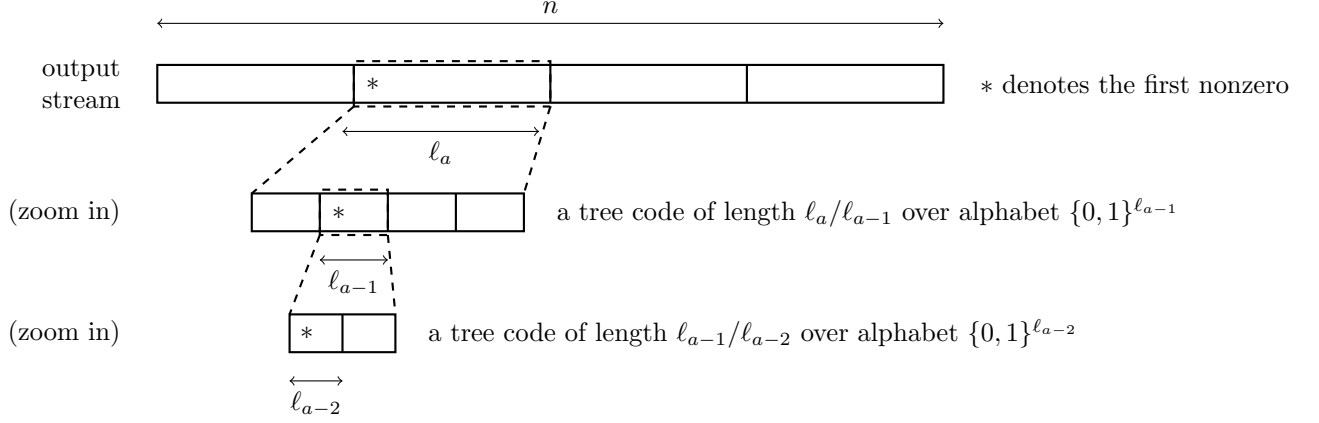


Figure 2: Schematic view of the multi-scale guarantee, for some $a \in [L]$. The symbol $*$ marks the location of the first nonzero input. The first nonzero ℓ_a -block contains a smaller tree code.

2.2.1 A multi-scale MDS tree code

We now construct the central object used in the binary construction. The goal is to build a single online linear map

$$\mathcal{R} : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

which simultaneously behaves like an MDS tree code at several block scales. At scale a , the input is viewed as blocks of length ℓ_{a+1} , each consisting of $m_a = \ell_{a+1}/\ell_a$ subblocks of length ℓ_a . The guarantee is that, within the first ℓ_{a+1} -block where the input is nonzero, the pair

$$(x, \mathcal{R}(x))$$

satisfies the MDS guarantee with respect to its ℓ_a -subblocks.

Proposition 2.4 (A tree code which is MDS at many scales altogether). *Let*

$$\ell_0 < \ell_1 < \dots < \ell_L < \ell_{L+1} = n$$

be integers such that $\ell_a \mid \ell_{a+1}$ for every $a \in \{0, \dots, L\}$. For every $a \in \{0, \dots, L\}$, write

$$m_a = \frac{\ell_{a+1}}{\ell_a}.$$

Assume that for every $a \in \{0, \dots, L\}$ we are given the redundancy map

$$\mathcal{R}^a : (\{0, 1\}^{\ell_a})^{m_a} \rightarrow (\{0, 1\}^{\ell_a})^{m_a}$$

of a normal $\mathbb{F}_2$ -linear MDS tree code of length m_a over alphabet $\{0, 1\}^{\ell_a}$.

Then there exists an explicit online $\mathbb{F}_2$ -linear map

$$\mathcal{R} : \{0, 1\}^n \rightarrow \{0, 1\}^n$$

with the following property. Fix a scale $a \in \{0, \dots, L\}$. Decompose $[n]$ into n/ℓ_{a+1} consecutive blocks of length ℓ_{a+1} ,

$$[n] = Q_1^a \cup \dots \cup Q_{n/\ell_{a+1}}^a.$$

Write

$$Q_u^a = P_{u,1}^a \cup P_{u,2}^a \cup \dots \cup P_{u,m_a}^a,$$

where $P_{u,1}^a, \dots, P_{u,m_a}^a$ are the consecutive ℓ_a -subblocks of Q_u^a .

Then for every nonzero $x \in \{0,1\}^n$, the following holds. Let

$$(u, s) \in \left[\frac{n}{\ell_{a+1}} \right] \times [m_a]$$

be minimal in lexicographic order such that

$$x_{P_{u,s}^a} \neq \bar{0}.$$

Then for every suffix length $h \in [m_a - s + 1]$,

$$\left| \left\{ j \in [s, s + h - 1] : x_{P_{u,j}^a} \neq \bar{0} \right\} \right| + \left| \left\{ j \in [s, s + h - 1] : \mathcal{R}(x)_{P_{u,j}^a} \neq \bar{0} \right\} \right| > h. \quad (2.2)$$

Proof. Given the maps $\mathcal{R}^0, \dots, \mathcal{R}^L$, we define new maps

$$\mathcal{R}^{(a)} : (\{0,1\}^{\ell_a})^{m_a} \rightarrow (\{0,1\}^{\ell_a})^{m_a}$$

recursively, as follows. For $a = 0$, set

$$\mathcal{R}^{(0)} = \mathcal{R}^0.$$

For $a > 0$, on input

$$y = (y_1, \dots, y_{m_a}) \in (\{0,1\}^{\ell_a})^{m_a},$$

where each $y_j \in \{0,1\}^{\ell_a}$ is viewed as a word over $\{0,1\}^{\ell_{a-1}}$, define

$$\mathcal{R}^{(a)}(y) := \mathcal{R}^a(\mathcal{R}^{(a-1)}(y_1), \dots, \mathcal{R}^{(a-1)}(y_{m_a})).$$

Finally, to define the desired map $\mathcal{R}$, view each $x \in \{0,1\}^n$ as an element of $(\{0,1\}^{\ell_L})^{m_L}$, and set

$$\mathcal{R}(x) := \mathcal{R}^{(L)}(x).$$

The recursive definition is bitwise online: this follows from the normality of the maps $\mathcal{R}^a$, together with the induction hypothesis that $\mathcal{R}^{(a-1)}$ is bitwise online. Hence $\mathcal{R}$ is online.

It remains to see that Equation (2.2) is satisfied. We prove the following stronger statement by induction on a : the map $\mathcal{R}^{(a)}$ satisfies Equation (2.2) at every scale $0, \dots, a$ inside its ℓ_{a+1} -bit input.

For $a = 0$, this is exactly the MDS property of $\mathcal{R}^0$. Assume the claim for $a - 1$, and let

$$y = (y_1, \dots, y_{m_a}) \in (\{0,1\}^{\ell_a})^{m_a}.$$

For every $j \in [m_a]$, the induction hypothesis applied to y_j and to a suffix of length 1 implies that

$$y_j \neq \bar{0} \iff \mathcal{R}^{(a-1)}(y_j) \neq \bar{0}.$$

Thus the word

$$z := (\mathcal{R}^{(a-1)}(y_1), \dots, \mathcal{R}^{(a-1)}(y_{m_a}))$$

has the same zero pattern as y . Applying the MDS property of $\mathcal{R}^a$ to z therefore gives Equation (2.2) at scale a .

It remains only to note that the smaller scales are preserved. Let s be the first index with $y_s \neq \bar{0}$. Since $z_j = \bar{0}$ for all $j < s$, and since $\mathcal{R}^a$ is normal, its s -th output block on input z is exactly

$$z_s = \mathcal{R}^{(a-1)}(y_s).$$

Thus, inside the first nonzero ℓ_a -block y_s , the final output of $\mathcal{R}^{(a)}$ coincides with the output of $\mathcal{R}^{(a-1)}$. Therefore all guarantees at scales $0, \dots, a-1$ follow from the induction hypothesis inside y_s . This completes the induction. Taking $a = L$ proves Equation (2.2) for $\mathcal{R}$. $\square$

2.2.2 A ρ -schedule

The binary construction will need to schedule error correcting code information which is output by different scales. We formalize the scheduling object that we will use.

We make the following definition, of a ρ -schedule for $\rho \in (0, 1)$, which is a way to allocate the n time-points of S work-slots, such that for every task $a \in \{0, \dots, L\}$ associated with a time-length $\ell_a \in \mathbb{N}$, on each time between every two consecutive multiples of ℓ_a , the schedule allocates $\rho\ell_a$ work-points indexed by $\{1, \dots, \rho\ell_a\}$, for a , on some slot; the schedule never allocates work for a on more than slot at a time. More formally,

Definition 2.5. *Given integers $0 < \ell_0 < \dots < \ell_L$, and $S \in \mathbb{N}$, we say that a function $\pi : [n] \times [S] \rightarrow (\{0, \dots, L\} \times \mathbb{N}) \cup \{\perp\}$ is a ρ -schedule if the following holds.*

1. *For every $a \in \{0, \dots, L\}$ and $t \in [n]$ there is at most one $r \in [S]$ such that $\pi(t, r) = (a, \cdot)$.*
2. *For every $a \in \{0, \dots, L\}$ and $t \in [\ell_a]$, for every $r \in [S]$, $\pi(t, r) \neq (a, \cdot)$.*
3. *For every $a \in \{0, \dots, L\}$ and $f \in [\frac{n}{\ell_a} - 1]$, $\{\pi(t, r) \mid t \in \{f\ell_a + 1, \dots, (f+1)\ell_a\}, r \in [S]\} \cap (\{a\} \times \mathbb{N}) = \{a\} \times [\rho\ell_a]$.*

The next lemma shows that such a schedule exists with few slots.

Lemma 2.6. *Let*

$$0 < \ell_0 \mid \ell_1 \mid \dots \mid \ell_L$$

be integers, and let $0 < \rho \leq 1/2$ be such that $\rho\ell_a \in \mathbb{N}$ for every $a \in \{0, \dots, L\}$. Then there is an explicit ρ -schedule with

$$S = \left\lceil \frac{L+1}{\lfloor 1/\rho \rfloor} \right\rceil \leq \lceil 2\rho(L+1) \rceil.$$

Proof. Let

$$k = \left\lfloor \frac{1}{\rho} \right\rfloor.$$

Partition the scales $0, \dots, L$ into consecutive groups of size at most k , and assign one slot to each group. Thus the number of slots is

$$S = \left\lceil \frac{L+1}{k} \right\rceil = \left\lceil \frac{L+1}{\lfloor 1/\rho \rfloor} \right\rceil.$$

We identify $[n]$ with the time steps of the schedule. For each $a \in \{0, \dots, L\}$ and each $f \in [n/\ell_a - 1]$, a task of scale a becomes available at time $f\ell_a + 1$. This task consists of $\rho\ell_a$ units, indexed by

$$1, \dots, \rho\ell_a,$$

and must be processed during the interval

$$\{f\ell_a + 1, \dots, (f+1)\ell_a\}.$$

Within each group, at every time $t \in [n]$, the corresponding slot processes one unit from an available unfinished task of *smallest* scale index a in the group.⁴ If there is no available unfinished task in the group, the slot is idle.

We define $\pi(t, r) = (a, i)$ if at time t , slot r processes the i -th unit of a task of scale a , and define $\pi(t, r) = \perp$ if the slot is idle. By construction, the schedule never allocates work for the same scale on more than one slot at the same time.

It remains to prove that every task is completed by the end of its interval. Fix a group and consider a task J of scale a , becoming available at time

$$T + 1, \quad \text{where } T = q\ell_a,$$

and with deadline $T + \ell_a$. During the interval

$$I = \{T + 1, \dots, T + \ell_a\},$$

only tasks of smaller-index scales in the same group can be processed before J , assuming the claim has already been proved for earlier tasks.

For each smaller-index scale b in the group, the divisibility $\ell_b \mid \ell_a$ implies that scale b has exactly ℓ_a/ℓ_b tasks whose intervals are contained in I . These tasks contain in total

$$\frac{\ell_a}{\ell_b} \cdot \rho\ell_b = \rho\ell_a$$

units of work. Since there are at most $k - 1$ such smaller-index scales, the total amount of work that can be processed before J in I is at most

$$(k - 1)\rho\ell_a \leq (1 - \rho)\ell_a,$$

where the last inequality uses $k \leq 1/\rho$. Therefore at least $\rho\ell_a$ time steps of the group's slot remain for J , which is enough to complete all its units. Thus every task is completed by the end of its interval. $\square$

2.2.3 The main construction

We will use the following standard fact about explicit binary linear codes.

Fact 2.7. *For every $0 < \Delta < 1/2$, there exists $\rho = \rho(\Delta) = O(H(\Delta))$ such that, for every $k \in \mathbb{N}$, there is an explicit systematic binary linear code of dimension k , length at most $(1 + \rho)k$, and relative distance at least Δ . Equivalently, the encoding has the form*

$$x \mapsto (x, R_k^{\text{bin}}(x)),$$

where

$$R_k^{\text{bin}} : \{0, 1\}^k \rightarrow \{0, 1\}^{r_k}$$

is linear and $r_k \leq \rho k$.

⁴A naive greedy construction which always chooses the smallest available scale globally, without partitioning into groups, fails.

With this we are ready to prove the theorem of this section.

Theorem 2.8 (A binary tree code with improved parameters). *For every $0 < \Delta < 1/2$ and every $n' \in \mathbb{N}$, there is $\tilde{n} \geq n'$ such that there is an explicit $\mathbb{F}_2$ -linear binary tree code*

$$\widetilde{\mathcal{TC}} : \{0, 1\}^{\tilde{n}} \rightarrow (\{0, 1\}^r)^{\tilde{n}}$$

with distance $\Omega(\Delta)$, where

$$r = O(H(\Delta) \log \log \tilde{n}).$$

Before proving the theorem, we describe the encoding of the claimed tree code. We first describe an intermediate encoder $\mathcal{TC}$ in Figure 3, and state the key lemma used in its analysis, Lemma 2.9. We then describe the final encoder $\widetilde{\mathcal{TC}}$ in Figure 4, which consists of two relatively shifted copies of the intermediate encoder. We conclude with the proof of the theorem.

Algorithm 1: the encoder $\mathcal{TC}$.

Parameters.

- Let $0 < \ell_0 < \ell_1 < \dots < \ell_L < \ell_{L+1} = n$ be integers such that $\ell_a \mid \ell_{a+1}$.
- Let $\rho \in (0, 1)$ be such that for every a , $\rho \ell_a \in \mathbb{N}$.

Building Blocks.

- Let $\mathcal{R} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be the (redundancy map of the) multi-scale block tree-code from [Proposition 2.4](#).
- $\pi : [n] \times [S] \rightarrow (\{0, \dots, L\} \times \mathbb{N}) \cup \{\perp\}$ the ρ -schedule from [Lemma 2.6](#) with $S = \left\lceil \frac{L+1}{[1/\rho]} \right\rceil$.
- For every $a \in \{0, \dots, L\}$, let $R_{\ell_a}^{\text{bin}} : \{0, 1\}^{\ell_a} \rightarrow \{0, 1\}^{\rho \ell_a}$ be the map from [Fact 2.7](#).

The Encoding.

On input $x \in \{0, 1\}^n$, the encoding $\mathcal{TC} : \{0, 1\}^n \rightarrow (\{0, 1\}^{2+2S})^n$ is defined as follows.

1. For every $a \in \{0, \dots, L\}$ and $j \in [\frac{n}{\ell_a}]$, define

$$\begin{aligned} E_{a,j} &:= R_{\ell_a}^{\text{bin}}(x_{(j-1)\ell_a+1, \dots, j\ell_a}) \in \{0, 1\}^{\rho \ell_a}, \\ O_{a,j} &:= R_{\ell_a}^{\text{bin}}(\mathcal{R}(x)_{(j-1)\ell_a+1, \dots, j\ell_a}) \in \{0, 1\}^{\rho \ell_a}. \end{aligned}$$

(In words: $E_{a,j}$ is the binary-redundancy of the j -th block of x , within the scale a ; $O_{a,j}$ is the binary-redundancy of the j -th block of $\mathcal{R}(x)$, within the scale a .)

2. For each slot number $r \in [S]$ define two strings $e^r = (e_1^r, \dots, e_n^r) \in \{0, 1\}^n$ and $o^r = (o_1^r, \dots, o_n^r) \in \{0, 1\}^n$, as follows. For $t \in [n]$,

$$\begin{aligned} e_t^r &= \begin{cases} 0 & \text{if } \pi(t, r) = \perp, \\ (E_{a, \lceil \frac{t}{\ell_a} \rceil - 1})_h & \text{if } \pi(t, r) = (a, h), \end{cases} \\ o_t^r &= \begin{cases} 0 & \text{if } \pi(t, r) = \perp, \\ (O_{a, \lceil \frac{t}{\ell_a} \rceil - 1})_h & \text{if } \pi(t, r) = (a, h). \end{cases} \end{aligned}$$

(I.e., e^r and o^r are the ordering of the redundancy-blocks $\{E_{a,j}\}$ and $\{O_{a,j}\}$ according to the r -th slot of the schedule π : if $\pi(t, r) = (a, h)$, where t is within the j -th ℓ_a -sized interval of $[n]$, then we set the t -th location in e^r and o^r to be the h -th element of $E_{a,j-1}$ and the h -th element of $O_{a,j-1}$, accordingly.)

3. At each time $t \in [n]$, set the t -th output to

$$\mathcal{TC}(x)_t = (x_t, \mathcal{R}(x)_t, e_t^1, \dots, e_t^S, o_t^1, \dots, o_t^S).$$

(That is, each output symbol is composed of the current bit of x , the current bit of $\mathcal{R}(x)$, and one bit for each of the $2S$ binary-redundancy strings $\{e^r\}_{r \in [S]}, \{o^r\}_{r \in [S]}$.)

Figure 3: An intermediate encoder $\mathcal{TC}$, used by the final encoder $\widetilde{\mathcal{TC}}$ described next.

The following lemma is for a specific scale a^* . It shows that in case that a suffix starting from the split s is contained in an interval which ends before the next time that an ℓ_{a^*+1} -block of the multi-scale tree-code is reached, then the described encoding $\mathcal{TC}$ assures high-distance, proportional to the suffix length.

Lemma 2.9. *Let $\mathcal{TC}$ be the encoder defined in [Figure 3](#), and fix $a^* \in \{0, \dots, L\}$. Let $x \in \{0, 1\}^n$*

be non-zero, and let $s := \min\{t \in [n] \mid x_t \neq 0\}$. Further assume that $h \in [n - s + 1]$ is such that $h \geq 2\ell_{a^*}$, and moreover, for some $d \in \mathbb{N}$, $I := [s, s + h - 1] \subseteq [(d - 1)\ell_{a^*+1} + 1, d\ell_{a^*+1}]$. Then $\text{HammingWeight}(\mathcal{TC}(x)_I) \geq \Omega(\Delta h)$.

Proof. Let $J_1, \dots, J_q$ be the consecutive ℓ_{a^*} blocks whose right endpoints are contained in I , that is $\{J_1, \dots, J_q\} = \{[(f - 1)\ell_{a^*} + 1, f\ell_{a^*}] \mid f \in \mathbb{N}, f\ell_{a^*} \in I\}$. Note that

$$q \geq \lfloor \frac{h}{\ell_{a^*}} \rfloor, \quad (2.3)$$

and therefore by the assumption $h \geq 2\ell_{a^*}$, $q \geq 2$. Recall that the t -th output, $\mathcal{TC}(x)_t$, is a tuple of $2 + 2S$ elements, the first two being x_t and $\mathcal{R}(x)_t$, and then $\hat{e}_t := (e_t^1, \dots, e_t^S)$ and $\hat{o}_t := (o_t^1, \dots, o_t^S)$ which consist of the remaining $2S$ elements of the output. We have, by [Proposition 2.4](#), that

$$|\{j \in [q - 1] : x_{J_j} \neq 0\}| + |\{j \in [q - 1] : \mathcal{R}(x)_{J_j} \neq 0\}| > q - 1.$$

Therefore either

$$|\{j \in [q - 1] : x_{J_j} \neq 0\}| \geq \frac{q - 1}{2} \quad \text{or} \quad |\{j \in [q - 1] : \mathcal{R}(x)_{J_j} \neq 0\}| \geq \frac{q - 1}{2}.$$

Assume the first. Observe that for every $J_j = [(f - 1)\ell_{a^*} + 1, f\ell_{a^*}]$ for $j \in [q - 1]$, $R_{\ell_{a^*}}^{\text{bin}}(x_{J_j})$ is embedded in $\hat{e}_{J_{j+1}}$ where $J_{j+1} \subseteq I$. Therefore, if $x_{J_j} \neq 0$, then

$$|\{t \in [|J_j|] \mid (x_{J_j})_t \neq 0\}| + |\{t \in [|J_{j+1}|] \mid (\hat{e}_{J_{j+1}})_t \neq 0\}| \geq \Delta\ell_{a^*},$$

by the $R_{\ell_{a^*}}^{\text{bin}}$ distance guarantee in [Fact 2.7](#). We conclude that

$$|\{t \in I \mid x_t \neq 0\}| + |\{t \in I \mid \hat{e}_t \neq 0\}| \geq \frac{q - 1}{2} \cdot \Delta\ell_{a^*},$$

and thus that

$$\text{HammingWeight}(\mathcal{TC}(x)_I) \geq \frac{q - 1}{4} \cdot \Delta\ell_{a^*} = \Omega(q\Delta\ell_{a^*}),$$

recalling that $q \geq 2$. Plugging in [Equation \(2.3\)](#) we get that $\text{HammingWeight}(\mathcal{TC}(x)_I) \geq \Omega(h\Delta)$, as desired. The second case, that $|\{j \in [q - 1] : \mathcal{R}(x)_{J_j} \neq 0\}| \geq \frac{q - 1}{2}$, has an identical analysis, considering $\mathcal{R}(x)$ instead of x and $\hat{o}$ instead of $\hat{e}$. $\square$

We now state the final encoding used, followed by the proof for [Theorem 2.8](#).

Algorithm 2: the encoder $\widetilde{\mathcal{TC}}$.**Parameters.**

- Let $0 < \ell_0 < \ell_1 < \dots < \ell_L < \ell_{L+1} = n$ be integers such that $\ell_a \mid \ell_{a+1}$, and such that

$$\sigma = \sum_{a=1}^L \frac{\ell_a}{2} \in \mathbb{N}. \quad (2.4)$$

- Let $\rho \in (0, 1)$ be such that for every a , $\rho \ell_a \in \mathbb{N}$.

Building Blocks.

- Let $\mathcal{TC} : \{0, 1\}^n \rightarrow (\{0, 1\}^{2+2S})^n$ be the encoding defined in Figure 3, by the same parameters, where $S = \left\lceil \frac{L+1}{\lfloor 1/\rho \rfloor} \right\rceil$.

The Encoding.

On input $x \in \{0, 1\}^{n-\sigma}$, the encoding $\widetilde{\mathcal{TC}} : \{0, 1\}^{n-\sigma} \rightarrow (\{0, 1\}^{4+4S})^{n-\sigma}$ is defined as follows.

- At each time $t \in [n - \sigma]$, set the t -th output to

$$\widetilde{\mathcal{TC}}(x)_t = (\mathcal{TC}(x \circ 0^\sigma)_t, \mathcal{TC}(0^\sigma \circ x)_{t+\sigma}),$$

where $0^\sigma := \underbrace{(0, \dots, 0)}_{\sigma \text{ times}}.$

Figure 4: The main encoder $\widetilde{\mathcal{TC}}$, used for proving Theorem 2.8.

Proof for Theorem 2.8.

Setting parameters. Our aim is to construct $\widetilde{\mathcal{TC}}$ by relying on Proposition 2.4, which, recall, constructs a “multi-scale” tree code, of different alphabet sizes, corresponding to the scales $\ell_0, \dots, \ell_{L+1} \in \mathbb{N}$. Before invoking Proposition 2.4, we start the proof by choosing these scales. Towards, we first set intermediate values $e_0, \dots, e_{L+1}$ by setting $e_0 = 9$, and defining recursively $e_{a+1} = e_a + \lfloor \frac{e_a}{3} \rfloor$. Then, we set for every a , $\ell_a = 2^{e_a}$. We set L to be the smallest integer for which

$$\ell_{L+1} \geq 2n'.$$

Observe that for every $a \in \{0, \dots, L\}$

$$\ell_{a+1} \geq 8\ell_a \quad (2.5)$$

and $\ell_a \mid \ell_{a+1}$, as both are powers of two. The a -th level depth is

$$m_a = \frac{\ell_{a+1}}{\ell_a} = 2^{e_{a+1}-e_a}; \quad \implies m_a^3 = 2^{3(e_{a+1}-e_a)} = 2^{3\lfloor e_a/3 \rfloor} \leq 2^{e_a} = \ell_a,$$

that is, the alphabet length ℓ_a is large enough to apply Pudlák’s MDS Theorem 1.6 tree code of length m_a at scale a .

Further, we define

$$\begin{aligned} n &:= \ell_{L+1}, \\ \sigma &:= \sum_{a=1}^L \frac{\ell_a}{2}, \\ \tilde{n} &:= n - \sigma \end{aligned} \quad (2.6)$$

where $\tilde{n}$ is to be the constructed code length. Observe that $\sigma < \frac{n}{4}$ and so

$$\tilde{n} > \frac{3}{4}n \geq \frac{6}{4}n' > n'. \quad (2.7)$$

Finally, as the sequence e_a grows exponentially in a , $L = O(\log e_{L+1}) = O(\log \log \tilde{n})$, since $e_{L+1} = \log n = O(\log \tilde{n})$ by Equation (2.7).

The encoding. Let $\rho = \rho(\Delta) = O(H(\Delta))$ be as in fact Fact 2.7. By decreasing Δ by a constant factor, we may assume that $\rho \leq 1/2$, and that $\rho = 2^{-b}$ for some $b \in \mathbb{N}$.⁵ Let $\widetilde{\mathcal{TC}}$ be the encoder described in Figure 4. We argue that this encoder satisfies the desired properties.

Correctness. The encoder is online since $\mathcal{R}$ is online by Proposition 2.4, and every error correcting code bit, a result of $R_{\ell_a}^{\text{bin}}$, is used only after its input block has been fully read. It is $\mathbb{F}_2$ -linear since $\mathcal{R}$ and the maps $R_{\ell_a}^{\text{bin}}$ are linear.

Rate analysis. By the definition of the encoder, the output alphabet is $\{0, 1\}^r$, where

$$r = 2(2 + 2S) = 4 + 4S,$$

where by Lemma 2.6,

$$S = \left\lceil \frac{L+1}{\lceil 1/\rho \rceil} \right\rceil = O(\rho L).$$

Since $L = O(\log \log \tilde{n})$ and $\rho = O(H(\Delta))$, the rate is

$$\Omega\left(\frac{1}{H(\Delta) \log \log \tilde{n}}\right).$$

Distance analysis. It remains to prove the distance guarantee. Since $\widetilde{\mathcal{TC}}$ is linear, it suffices to consider a single non-zero input $x \in \{0, 1\}^{\tilde{n}}$. Let

$$s := \min\{t \in [\tilde{n}] \mid x_t \neq 0\}$$

be the split, and fix a suffix length $h \in [\tilde{n} - s + 1]$. We will lower bound the number of non-zero output symbols in the interval

$$I := [s, s + h - 1].$$

Towards that, we first observe that if $h < 2\ell_0$ then the fraction of non-zeroes within the interval is larger than $\frac{1}{2\ell_0} = \Omega(1)$, and so we proceed by assuming that $h \geq 2\ell_0$. We define

$$a^* := \max\{a \in \{0, \dots, L\} \mid 2\ell_a \leq h\}.$$

Note that this choice guarantees that within the output stream of a multi-scale tree code, at least two ℓ_{a^*} -size blocks end within I . We further notate

$$m^* := \frac{\ell_{a^*+1}}{\ell_{a^*}},$$

⁵We use this only to ensure that $\rho\ell_a \in \mathbb{N}$ whenever there is binary redundancy to schedule: $\rho\ell_a$ is the length of the additional binary redundancy scheduled at scale a . If $\rho\ell_a < 1$, then there is no such redundancy to schedule.

the length of a tree-code with block size ℓ_{a^*} .

Consider the two parts of $\widetilde{\mathcal{TC}}(x)$. The first corresponds to first $\tilde{n}$ locations of $\mathcal{TC}(x \circ 0^\sigma) := y^{(0)}$, and the second corresponds to $\mathcal{TC}(0^\sigma \circ x)_{\{\sigma+1, \dots, n\}} := y^{(1)}$. We consider two points in $y^{(0)}$ and $y^{(1)}$,

$$r^{(0)} := \ell_{a^*+1} \cdot \lceil \frac{s}{\ell_{a^*+1}} \rceil \in [n] \quad \text{and} \quad r^{(1)} := \ell_{a^*+1} \cdot \lceil \frac{\sigma + s}{\ell_{a^*+1}} \rceil - \sigma \in [\tilde{n}].$$

Observe that $r^{(0)}$ is the point in $y^{(0)}$ where the tree code with block length ℓ_{a^*} , which contains s , ends. Similarly, $r^{(1)}$ is the point in $y^{(1)}$ where the tree code with block length ℓ_{a^*} , which contains s , ends. Between $y^{(0)}$ and $y^{(1)}$, let $\eta \in \{0, 1\}$ correspond to the one for which $r^{(\eta)} - s + 1$ is maximal; observe that $r^{(\eta)} - s + 1$ is the time length from the split to the tree code end within $y^{(\eta)}$, and we claim that it must be large. To show that, we write $\sigma = \sigma' \ell_{a^*+1} + (\sigma \bmod \ell_{a^*+1})$ for an integer σ' and observe that $r^{(1)} = \ell_{a^*+1} \cdot \lceil \frac{(\sigma \bmod \ell_{a^*+1}) + s}{\ell_{a^*+1}} \rceil - (\sigma \bmod \ell_{a^*+1})$ and that adding $(\sigma \bmod \ell_{a^*+1})$ in the numerator increases the ceiling by at most 1. Therefore,

$$\begin{aligned} |r^{(1)} - r^{(0)}| &= \left| \ell_{a^*+1} \left(\lceil \frac{(\sigma \bmod \ell_{a^*+1}) + s}{\ell_{a^*+1}} \rceil - \lceil \frac{s}{\ell_{a^*+1}} \rceil \right) - (\sigma \bmod \ell_{a^*+1}) \right| \\ &\geq \min((\sigma \bmod \ell_{a^*+1}), \ell_{a^*+1} - (\sigma \bmod \ell_{a^*+1})). \end{aligned}$$

It follows that $r^{(\eta)} - s + 1 \geq \min((\sigma \bmod \ell_{a^*+1}), \ell_{a^*+1} - (\sigma \bmod \ell_{a^*+1}))$. Now, by σ 's definition in Equation (2.6), and using the fact that for every a , $\ell_a < \ell_{a+1}$ are powers of 2, then if $a^* < L$

$$\sigma \bmod \ell_{a^*+1} = \left(\sum_{a=1}^{a^*+1} \frac{\ell_a}{2} \right) \bmod \ell_{a^*+1}.$$

Note that $\sum_{a=1}^{a^*} \ell_a < \frac{1}{2} \ell_{a^*+1}$ by Equation (2.5), and therefore $\sum_{a=1}^{a^*+1} \frac{\ell_a}{2} < \frac{3}{4} \ell_{a^*+1}$. We conclude that if $a^* < L$

$$r^{(\eta)} - s + 1 > \frac{1}{4} \ell_{a^*+1}, \tag{2.8}$$

and so we have successfully bounded the time length from the split to the tree code end within $y^{(\eta)}$.

In order to bound the non-zeroes in $\mathcal{TC}(x)$, we will only consider the non-zeroes within $y^{(\eta)}$, and so we proceed by bounding the latter. Towards that, we split into cases.

1. The first case is that $r^{(\eta)} \geq s + h - 1$. In this case, by Lemma 2.9 with regard to a suffix of length h starting from the split, applied on $\mathcal{TC}(x \circ 0^\sigma)$ in case that $\eta = 0$, or on $\mathcal{TC}(0^\sigma \circ x)$ in case $\eta = 1$ (in either case both outputs are included in $\widetilde{\mathcal{TC}}(x)$), yields $\text{HammingWeight}(\widetilde{\mathcal{TC}}(x)_I) = \Omega(\Delta h)$.
2. The second case is that $r^{(\eta)} < s + h - 1$. In this case we define $h' := r^{(\eta)} - s + 1$, and consider the interval $I' := [s, s + h' - 1] = [s, r^{(\eta)}] \subseteq I$. We again apply Lemma 2.9, this time with regards to the suffix of length h' starting from the split (again on either $\mathcal{TC}(x \circ 0^\sigma)$ or on $\mathcal{TC}(0^\sigma \circ x)$, dependent on η). Notice that by Equation (2.8) and Equation (2.5), indeed we have $h' \geq 2\ell_{a^*}$ as required to apply the lemma. Observe that to use Equation (2.5) we need that $a^* < L$. If it was the case that $a^* = L$ then $r^{(\eta)} \geq \tilde{n}$, and thus since we are in the case that $r^{(\eta)} < s + h - 1$, this cannot be. Hence, applying Lemma 2.9 yields that $\text{HammingWeight}(\widetilde{\mathcal{TC}}(x)_{I'}) = \Omega(\Delta h') = \Omega(\Delta \ell_{a^*+1})$. Now, by the definition of a^* , $2\ell_{a^*+1} > h$. Thus we conclude that $\text{HammingWeight}(\widetilde{\mathcal{TC}}(x)_I) \geq \text{HammingWeight}(\widetilde{\mathcal{TC}}(x)_{I'}) = \Omega(\Delta h)$ in this case as well.

This proves the tree distance lower bound. $\square$

Remark 2.10 (Removing the entropy loss). *As stated, the $H(\Delta)$ factor in Theorem 2.8 can be replaced by Δ . The only place where the construction above loses a factor $O(H(\Delta))$, rather than $O(\Delta)$, is in the use of the binary error-correcting codes from Fact 2.7. This loss can be removed by first carrying out the construction over a polynomial-size alphabet.*

Indeed, take the basic alphabet to be $\{0,1\}^{a_0}$, with $a_0 = O(\log n)$, rather than a constant-size alphabet. The construction is otherwise the same, except that the auxiliary error-correcting codes are now applied over the alphabet $\{0,1\}^{a_0}$. Over such alphabets, one can use standard systematic codes whose redundancy has alphabet exponent $O(\Delta)$, rather than $O(H(\Delta))$. The same argument as in the proof of Theorem 2.8 then applies with $\rho = O(\Delta)$. Since the final rate obtained there is $\Omega(1/(\rho \log \log n))$, this gives rate

$$\Omega\left(\frac{1}{\Delta \log \log n}\right)$$

and distance $\Omega(\Delta)$.

Finally, applying the standard reduction from polynomial-size alphabets to the binary alphabet, as in Fact 3.11 of [BYCY22], loses only constant factors in the distance and rate. This gives the stated binary construction.

3 MDS tree codes over a smaller alphabet

In this part we prove the following (variable-alphabet) MDS tree code theorem.

Theorem 3.1. *For every $\ell \in \mathbb{N}_{\geq 3}$, there exist alphabets*

$$\Gamma_1, \dots, \Gamma_\ell$$

with $|\Gamma_t| \leq 2^{\ell^2}$ for every $t \in [\ell]$, and an explicit linear tree code

$$\mathcal{TC}_\ell : \Gamma_1 \times \dots \times \Gamma_\ell \rightarrow (\Gamma_1 \times \Gamma_1) \times \dots \times (\Gamma_\ell \times \Gamma_\ell)$$

which is MDS. Namely, for every nonzero

$$x = (x_1, \dots, x_\ell) \in \Gamma_1 \times \dots \times \Gamma_\ell,$$

if

$$s = \min\{t \in [\ell] : x_t \neq 0\},$$

then for every $h \in [\ell - s + 1]$,

$$|\{t \in [s, s + h - 1] : x_t \neq 0\}| + |\{t \in [s, s + h - 1] : (\mathcal{TC}_\ell(x)_t)_2 \neq 0\}| > h, \quad (3.1)$$

where $(\mathcal{TC}_\ell(x)_t)_2$ denotes the second coordinate of $\mathcal{TC}_\ell(x)_t \in \Gamma_t \times \Gamma_t$. In particular, $\mathcal{TC}_\ell$ has rate $1/2$ and distance larger than $1/2$.

The construction is based on shifted sums of the input symbols, and we will prove its correctness by an inductive argument. The lengths of the alphabets are chosen so that, at time t , the newly read symbol x_t has exactly the same length as each one of the two entries of the t -th output pair; this length is determined by how far the previous input symbols have been shifted up to time t .

Before presenting the construction, we define the alphabet lengths and the shifted-sum operation used by the encoder. We then state the inductive lemma used to prove the distance guarantee.

Definition 3.2. *Given $\ell \in \mathbb{N}_{\geq 3}$ we define:*

Alphabets $\Gamma_1, \dots, \Gamma_\ell$. Set

$$n_1 := \binom{\ell-1}{2},$$

and for every $t \geq 2$, define

$$n_t := n_{t-1} + t - 2.$$

Equivalently,

$$n_t = \binom{\ell-1}{2} + \binom{t-1}{2}.$$

For every $t \in [\ell]$, set

$$\Gamma_t := \{0, 1\}^{n_t}.$$

The shift operator. Given $v = (v_1, \dots, v_n) \in \{0, 1\}^n$, $q \geq 0$, and $r \geq n + q$, define

$$\text{shift}_r(v, q) \in \{0, 1\}^r$$

by

$$\text{shift}_r(v, q) := (\underbrace{0, \dots, 0}_{r-n-q \text{ times}}, v_1, \dots, v_n, \underbrace{0, \dots, 0}_{q \text{ times}}).$$

That is, $\text{shift}_r(v, q)$ shifts v q places to the left and pads by zeroes on both sides so that the resulting vector has length r .

The Eval operator. For $t \in [\ell]$, define

$$\text{Eval}_t : \Gamma_1 \times \dots \times \Gamma_\ell \rightarrow \Gamma_t$$

by

$$\text{Eval}_t(x_1, \dots, x_\ell) := \sum_{i=1}^t \text{shift}_{n_t}(x_i, (i-1)(t-i)),$$

where the sum is over $\mathbb{F}_2$.⁶

The proof of distance of the construction is based on the following lemma.

Lemma 3.3. Let $x_1, \dots, x_\ell$, with $x_i \in \Gamma_i$, be not all zero, and set

$$\nu := |\{i \in [\ell] \mid x_i \neq \bar{0}\}|, \quad s := \min\{i \in [\ell] \mid x_i \neq \bar{0}\}.$$

Then

$$|\{t \in [s, \ell] \mid \text{Eval}_t(x_1, \dots, x_\ell) = \bar{0}\}| < \nu.$$

Before proving [Lemma 3.3](#), we use it to define the tree code and prove [Theorem 3.1](#).

⁶The operator Eval_t is well-defined. Indeed, in the i -th summand, the vector x_i has length n_i and is shifted by $q = (i-1)(t-i)$. Thus we need $n_i + q \leq n_t$. For $i < t$, the quantity $n_i + (i-1)(t-i)$ is maximized at $i = t-1$, and hence $n_i + (i-1)(t-i) \leq n_{t-1} + t - 2 = n_t$.

Proof of Theorem 3.1. Let $\ell \in \mathbb{N}_{\geq 3}$, and let $\Gamma_1, \dots, \Gamma_\ell$ be the alphabets from Definition 3.2. Define

$$\mathcal{TC}_\ell : \Gamma_1 \times \dots \times \Gamma_\ell \rightarrow (\Gamma_1 \times \Gamma_1) \times \dots \times (\Gamma_\ell \times \Gamma_\ell)$$

by

$$\mathcal{TC}_\ell(x_1, \dots, x_\ell)_t := (x_t, \text{Eval}_t(x_1, \dots, x_\ell)).$$

By an easy inspection of the Eval operator, this gives an online linear encoding. Moreover, for every $t \in [\ell]$,

$$n_t = \binom{\ell-1}{2} + \binom{t-1}{2} \leq 2 \binom{\ell-1}{2} < \ell^2,$$

and therefore

$$|\Gamma_t| = 2^{n_t} < 2^{\ell^2}.$$

The MDS guarantee Equation (3.1) follows immediately from Lemma 3.3. As for the “in particular” part, by the MDS guarantee, on every relevant interval of length h , strictly more than h nonzero output entries out of $2h$, thus the distance is larger than $1/2$. $\square$

It remains to prove Lemma 3.3. We prove a stronger statement, Lemma 3.6, which implies it, by induction. We first introduce two definitions.

Definition 3.4. For every $n \in \mathbb{N}$ and every nonzero $x \in \{0, 1\}^n$, define

$$\text{LeftMostNonzero}(x) := \min\{i \in [n] \mid x_i \neq 0\}.$$

Definition 3.5. For every not-all-zero $x_1, \dots, x_\ell$, with $x_i \in \Gamma_i$, we define, for every $t \geq \min\{i \in [\ell] \mid x_i \neq \bar{0}\}$,

$$\text{LeftMostNonzeroColumn}_t(x_1, \dots, x_\ell) := \min_{i \leq t : x_i \neq \bar{0}} \text{LeftMostNonzero}(\text{shift}_{n_t}(x_i, (i-1)(t-i))).$$

With these definitions in place, we now state and prove the stronger lemma.

Lemma 3.6. Let $x_1, \dots, x_\ell$, with $x_i \in \Gamma_i$, be not all zero, and set

$$I(x_1, \dots, x_\ell) := \{i \in [\ell] \mid x_i \neq \bar{0}\}, \quad \nu := |I(x_1, \dots, x_\ell)|.$$

Then the set

$$T(x_1, \dots, x_\ell) := \{t \in [\ell] \mid t \geq \min I(x_1, \dots, x_\ell) \wedge \text{Eval}_t(x_1, \dots, x_\ell)_{\text{LeftMostNonzeroColumn}_t(x_1, \dots, x_\ell)} = 0\}$$

has size less than ν .

Proof. The proof is by induction on ν .

If $\nu = 1$, let i^* be the unique index such that $x_{i^*} \neq \bar{0}$. Then for every $t \geq i^*$,

$$\text{Eval}_t(x_1, \dots, x_\ell) = \text{shift}_{n_t}(x_{i^*}, (i^* - 1)(t - i^*)).$$

Therefore the entry of $\text{Eval}_t(x_1, \dots, x_\ell)$ at $\text{LeftMostNonzeroColumn}_t(x_1, \dots, x_\ell)$ is nonzero. Hence $T(x_1, \dots, x_\ell) = \emptyset$, as desired.

Assume now that $\nu > 1$, and assume towards contradiction that $|T(x_1, \dots, x_\ell)| \geq \nu$. Let

$$t^* := \min T(x_1, \dots, x_\ell), \quad C^* := \text{LeftMostNonzeroColumn}_{t^*}(x_1, \dots, x_\ell).$$

Define

$$S := \{i \leq t^* \mid \text{shift}_{n_{t^*}}(x_i, (i-1)(t^* - i))_{C^*} \neq 0\}.$$

The set S is nonempty by the definition of C^* . Since $t^* \in T(x_1, \dots, x_\ell)$, we have

$$0 = \text{Eval}_{t^*}(x_1, \dots, x_\ell)_{C^*} = \sum_{i \in S} \text{shift}_{n_{t^*}}(x_i, (i-1)(t^* - i))_{C^*}.$$

The sum is over $\mathbb{F}_2$, and every summand is 1. Hence S has even size, so in particular $|S| \geq 2$. Let

$$i^* := \min S, \quad i^{**} := \min(S \setminus \{i^*\}).$$

Then $i^* < i^{**} \leq t^*$. Moreover,

$$\begin{aligned} \text{LeftMostNonzero}(\text{shift}_{n_{t^*}}(x_{i^*}, (i^* - 1)(t^* - i^*))) &= \\ \text{LeftMostNonzero}(\text{shift}_{n_{t^*}}(x_{i^{**}}, (i^{**} - 1)(t^* - i^{**}))) &= C^*. \end{aligned}$$

For every $t > t^*$, the row i^{**} moves farther to the left than the row i^* , since

$$(i^{**} - 1)(t - t^*) > (i^* - 1)(t - t^*).$$

Therefore

$$\begin{aligned} \text{LeftMostNonzero}(\text{shift}_{n_t}(x_{i^*}, (i^* - 1)(t - i^*))) &> \text{LeftMostNonzero}(\text{shift}_{n_t}(x_{i^{**}}, (i^{**} - 1)(t - i^{**}))) \\ &\geq \text{LeftMostNonzeroColumn}_t(x_1, \dots, x_\ell). \end{aligned}$$

In particular,

$$\text{LeftMostNonzero}(\text{shift}_{n_t}(x_{i^*}, (i^* - 1)(t - i^*))) \neq \text{LeftMostNonzeroColumn}_t(x_1, \dots, x_\ell) \quad (3.2)$$

for every $t > t^*$.

Now define $x'_1, \dots, x'_\ell$ by setting $x'_{i^*} = \bar{0}$ and $x'_i = x_i$ for every $i \neq i^*$. Then $x'_1, \dots, x'_\ell$ has exactly $\nu - 1$ nonzero entries. We claim that

$$T(x_1, \dots, x_\ell) \setminus \{t^*\} \subseteq T(x'_1, \dots, x'_\ell).$$

Indeed, let $t \in T(x_1, \dots, x_\ell) \setminus \{t^*\}$. Since $t^* = \min T(x_1, \dots, x_\ell)$, we have $t > t^*$. By Equation (3.2), removing x_{i^*} does not change the left-most nonzero column at time t . Hence

$$\text{LeftMostNonzeroColumn}_t(x'_1, \dots, x'_\ell) = \text{LeftMostNonzeroColumn}_t(x_1, \dots, x_\ell).$$

At this column, the contribution of x_{i^*} is zero, again by Equation (3.2). Therefore

$$\text{Eval}_t(x'_1, \dots, x'_\ell)_{\text{LeftMostNonzeroColumn}_t(x'_1, \dots, x'_\ell)} = \text{Eval}_t(x_1, \dots, x_\ell)_{\text{LeftMostNonzeroColumn}_t(x_1, \dots, x_\ell)} = 0.$$

Also $\min I(x'_1, \dots, x'_\ell) \leq i^{**} \leq t^* < t$, so $t \in T(x'_1, \dots, x'_\ell)$.

Thus

$$|T(x'_1, \dots, x'_\ell)| \geq |T(x_1, \dots, x_\ell)| - 1 \geq \nu - 1,$$

contradicting the induction hypothesis. This completes the proof. $\square$

Acknowledgment

The author thanks David Zuckerman for helpful discussions.

AI statement. No AI model was used in obtaining any of the ideas in this paper (the main ideas were obtained prior to the release of ChatGPT 5.6-sol). ChatGPT was locally used to generate the figures, as well as improving parts of the writing.

References

- [BH20] Siddharth Bhandari and Prahladh Harsha. A note on the explicit constructions of tree codes over polylogarithmic-sized alphabet. *arXiv preprint arXiv:2002.08231*, 2020.
- [Bra12] Mark Braverman. Towards deterministic tree code constructions. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 161–167. ACM, New York, 2012.
- [BYCN21] Inbar Ben Yaacov, Gil Cohen, and Anand Kumar Narayanan. Candidate Tree Codes via Pascal Determinant Cubes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, volume 207, pages 54:1–54:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- [BYCY22] Inbar Ben Yaacov, Gil Cohen, and Tal Yankovitz. Explicit binary tree codes with sub-logarithmic size alphabet. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 595–608, 2022.
- [CHS18] G. Cohen, B. Haeupler, and L.J. Schulman. Explicit binary tree codes with polylogarithmic size alphabet. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 535–544. ACM, 2018.
- [CSS25] Gil Cohen, Leonard J Schulman, and Piyush Srivastava. The rate-immediacy barrier in explicit tree code constructions. *arXiv preprint arXiv:2504.09388*, 2025.
- [GHK⁺16] R. Gelles, B. Haeupler, G. Kol, N. Ron-Zewi, and A. Wigderson. Towards optimal deterministic coding for interactive communication. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1922–1936, 2016.
- [MRR25] Tamer Mour, Alon Rosen, and Ron Rothblum. Locally testable tree codes. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5523–5559. SIAM, 2025.
- [MS14] Cristopher Moore and Leonard J. Schulman. Tree codes and a conjecture on exponential sums. In *ITCS’14—Proceedings of the 2014 Conference on Innovations in Theoretical Computer Science*, pages 145–153. ACM, New York, 2014.
- [NW19] A. K. Narayanan and M. Weidner. On decoding cohen-haeupler-schulman tree codes. *arXiv preprint arXiv:1909.07413*, 2019.
- [NW20] Anand Kumar Narayanan and Matthew Weidner. On decoding Cohen-Haeupler-Schulman tree codes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1337–1356. SIAM, 2020.
- [Pud16] P. Pudlák. Linear tree codes and the problem of explicit constructions. *Linear Algebra and its Applications*, 490:124–144, 2016.

- [Pud22] Pavel Pudlák. On matrices potentially useful for tree codes. *Information Processing Letters*, 174:106180, 2022.
- [Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In *Proceedings of the 25th annual ACM Symposium on Theory of Computing*, pages 747–756, 1993.
- [Sch94] Leonard J. Schulman. Postscript of 21 september 2003 to coding for interactive communication, 1994.
- [Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Trans. Inform. Theory*, 42(6, part 1):1745–1756, 1996. Codes and complexity.
- [Sho90] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of computation*, 54(189):435–447, 1990.
- [Yan25] Tal Yankovitz. Chs-alike $1/o(\log \log n)$ -rate tree codes from elementary binary shifts. In *Electronic Colloquium on Computational Complexity (ECCC): TR25-167*, 2025.

A Auxiliary proofs

Proof sketch for Theorem 2.2. Pudlák [Pud16, Section 6] constructs a lower-triangular matrix

$$W_n(x) \in \mathbb{F}_2[x]^{n \times n}$$

whose entries are

$$(W_n(x))_{i,j} = \begin{cases} x^{(n-i+j-1)^2}, & i \geq j, \\ 0, & i < j. \end{cases}$$

He proves that every square submatrix of $W_n(x)$ whose diagonal lies in the lower triangle has determinant equal to a nonzero polynomial in x . Every such determinant has degree at most

$$n(n-1)^2.$$

Indeed, each entry has degree at most $(n-1)^2$, and every determinant monomial is a product of at most n entries. Let

$$d = n(n-1)^2 + 1$$

and choose an irreducible polynomial $f(x) \in \mathbb{F}_2[x]$ of degree d . Such a polynomial can be found deterministically in polynomial time [Sho90]. Working over

$$\mathbb{F}_{2^d} \cong \mathbb{F}_2[x]/(f(x)),$$

and writing $\alpha = x \bmod f(x)$, we obtain the lower-triangular matrix

$$A_{i,j} = \begin{cases} \alpha^{(n-i+j-1)^2}, & i \geq j, \\ 0, & i < j. \end{cases}$$

Since no nonzero polynomial of degree smaller than d can vanish at α , every relevant minor of A is nonsingular. The arguments in [Pud16] then readily imply that the systematic encoding induced by A is an MDS tree code over $\mathbb{F}_{2^d}$.

Finally, fix an $\mathbb{F}_2$ -linear identification

$$\mathbb{F}_{2^d} \cong \mathbb{F}_2^d.$$

Under this identification, multiplication by each field element is an $\mathbb{F}_2$ -linear map, so the code becomes an $\mathbb{F}_2$ -linear map

$$(\{0, 1\}^d)^n \rightarrow (\{0, 1\}^d \times \{0, 1\}^d)^n.$$

Taking $\ell = d$ gives the claimed code. □