Really Complex Codes with Application to STARKs

Yuval Domb
yuval@ingonyama.com

Abstract

Reed-Solomon (RS) codes [RS60], representing evaluations of univariate polynomials over distinct domains, are fundamental in error correction and cryptographic protocols. Traditional RS codes leverage the Fourier domain for efficient encoding and decoding via Fast Fourier Transforms (FFT). However, in fields such as the Reals and some finite prime fields, limited root-of-unity orders restrict these methods. Recent research, particularly in the context of modern STARKs [BSBHR18b], has explored the Complex Fourier domain to construct real-valued RS codes, introducing novel transforms such as the Discrete Circle-Curve Transform (DCCT) for Real-to-Real transformations [HLP24]. Despite their efficiency, these transforms face limitations with high radix techniques and potentially other legacy know-how. In this paper, we propose a construction of Real-valued RS codes utilizing the Discrete Fourier Transform (DFT) over the Complex domain. This approach leverages well-established algebraic and Fourier properties to achieve an efficiency comparable to DCCT, while retaining compatibility with legacy techniques and optimizations.

1 Introduction

RS codewords are length-n sequences that represent evaluations of univariate polynomials of degree $k-1$ over a distinct domain, where $k < n$. Abundant RS codes utilize the Fourier domain, which is a cyclic group generated by a root-of-unity of suitable order, for their construction. This choice facilitates efficient encoding and decoding using FFT techniques [Bri88], especially effective for code lengths that are powers-of-two.

For power-of-two transform lengths, the root-of-unity must match this order. In certain fields like the Reals $\mathbb{R}$ and finite prime fields $\mathbb{F}_p$ where $p \equiv 3 \pmod 4$, there exists only one power-of-two root-of-unity, namely -1 of order 2, limiting options. However, Complex extensions of these fields ($\mathbb{R}^2$ and $\mathbb{F}_p^2$) offer roots-of-unity of higher, power-of-two orders.

Recent research, as discussed in [HLN23] and [HLP24], explores the use of the Complex Fourier domain for constructing Real-valued RS codes. In [HLP24], Haböck *et al.* introduced the DCCT, a novel non-Fourier approach specifically designed for Real-to-Real transformations. The DCCT employs trigonometric base functions, structured to satisfy the two-to-one decimation property (i.e. radix-2 butterflies), enabling efficient computational techniques. However, these same base functions limit the ability to apply higher radix techniques, which often provide computational and memory advantages [Bri88]. A key benefit of high-radix designs is the improved memory locality of twiddle factors. In such designs, the twiddle factors can be partitioned into two groups: those internal to the radix block and those external to it. Since the internal twiddle factors remain identical

across all radix blocks, they can be cached, offering a significant performance advantage, especially in massively parallel computing devices such as GPUs.

In this paper, we propose the Real Complex (RC) code, a novel error-correcting code derived from Real-valued RS codes in the Complex Fourier domain using the DFT [Dom23]. This approach leverages the well-established properties and computational advantages of Fourier transforms.

The paper is organized as follows. In Section 2, we introduce important notation and three key concepts: unique polynomial representation, Laurent polynomials, and Real-valued Discrete Fourier Transforms (RDFT). Section 3 covers the construction and properties of the new RC code. Section 4 focuses on practical aspects of working with RC codes, including Low Degree Extension (LDE) and polynomial operations. In Section 5, we explore the application of RC codes within STARKs. Finally, Section 6 discusses an optimized FFT algorithm for RDFT and compares RDFT with DCCT in the context of FFT, with particular emphasis on the 31-bit Mersenne prime field (M31).

2 Preliminaries

In this paper, notation is explicitly introduced when used, except where it has been previously defined. In the following, we outline a few general conventions that apply throughout.

The term "Complex field" refers to a quadratic extension of a given base field, which we call the "Real field." For example, the Complex field $\mathbb{C}_{\mathbb{F}} = \mathbb{F}^2$ represents a quadratic extension of the Real field $\mathbb{F}$, where $\mathbb{F}$ is an arbitrary field. The n-th root of unity in $\mathbb{C}_{\mathbb{F}}$ is commonly denoted by ω. Additionally, the concept of a two-dimensional unit-circle angle θ is extended to the Complex field $\mathbb{C}_{\mathbb{F}}$, enabling application of Euler's identity $\omega^\ell = \cos(\ell\theta) + \mathbf{i}\sin(\ell\theta)$ in this context [Bha11]. Note that we use boldface $\mathbf{i} \equiv \sqrt{-1}$, while i is used for indexing.

Polynomials are denoted in the usual way; however, to distinguish between standard polynomials and Laurent polynomials (see Subsection 2.2) in general usage, we denote standard polynomials by $P(x)$ and Laurent polynomials by $L(x)$.

2.1 Unique Polynomial Representation

Unique representation of univariate polynomials is as old as algebra itself [Bôc95]. We restate an appropriate version of it here as a theorem.

Theorem 2.1 (Unique Polynomial Representation (UPR))**.** *A univariate polynomial whose order is $n-1$ can be uniquely represented by any n distinct evaluations.*

Proof. Define a univariate polynomial of order $n-1$ over field K as

$$P(x) = \sum_{i=0}^{n-1} c_i x^i \tag{2.1}$$

where $c_i \in K$ are termed its coefficients. The translation from coefficients to n evaluations

can be represented by the following linear system

$$\begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^{n-1} \\ 1 & x_1 & x_1^2 & ... & x_1^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & ... & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} \tag{2.2}$$

where $\{x_i\}_{i\in[n]}$ is the evaluation domain. The translation matrix in (2.2) is a square Vandermonde matrix [Kal84] whose determinant is known to be

$$\prod_{\substack{i,j\in[n] \\ j>i}} x_j - x_i \tag{2.3}$$

For a distinct evaluation domain (i.e. $x_i \neq x_j \, \forall i \neq j$) the determinant (2.3) is necessarily non-zero and the linear system (2.2) is fully-determined. Hence, this system admits a unique solution. □

Definition 2.1 (RS Codeword)**.** *An RS codeword $\{P(x_i)\}_{i\in[n]} \in RS(n,k)$ with $k < n$ is the evaluation of an order $k-1$ polynomial $P(x)$ with coefficients $\{c_i\}_{i\in[k]}$ over a distinct length-n domain $\{x_i\}_{i\in[n]}$.*

Corollary 2.1.1. *Each codeword in $RS(n,k)$ has exactly $\binom{n}{k}$ determinations.*

Proof. By Theorem 2.1 any k distinct evaluations from $\{P(x_i)\}_{i\in[n]}$ uniquely determine $\{c_i\}_{i\in[k]}$. □

Remark 2.1.1. *RS codes are Maximum Distance Separable (MDS) codes, meaning they achieve the largest possible minimum distance for a given code of length n and dimension k, achieving the Singleton Bound $d_{min} \leq n - k + 1$. codes that achieve $d_{min} = n - k$ are often called "almost-MDS".*

2.2 Laurent Polynomials

Define a Laurent polynomial of order n over field K as the following univariate function

$$L(x) = \sum_{i=-\frac{n}{2}}^{\frac{n}{2}} c_i x^i \tag{2.4}$$

where n is even and $c_i \in K$ are termed its coefficients. Formally, the collection of Laurent polynomials forms an integral domain denoted by $K[x, x^{-1}]$ (see [Lan05]) with component-wise addition and multiplication governed by convolution of coefficient vectors (see Section 1.3 of [Dom23]).

Corollary 2.1.2. *Laurent polynomials obey the UPR property.*

Proof. Equation (2.4) can be rewritten as

$$L(x) = x^{-\frac{n}{2}} \sum_{i=0}^{n} c_{i-\frac{n}{2}} x^i \tag{2.5}$$

which leads to the Laurent version of (2.2)

$$\begin{bmatrix} L(x_0) \\ L(x_1) \\ \vdots \\ L(x_n) \end{bmatrix} = \begin{bmatrix} x_0^{-\frac{n}{2}} & & \\ & \ddots & \\ & & x_n^{-\frac{n}{2}} \end{bmatrix} \begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & ... & x_n^n \end{bmatrix} \begin{bmatrix} c_{-\frac{n}{2}} \\ c_{-\frac{n}{2}+1} \\ \vdots \\ c_{\frac{n}{2}} \end{bmatrix} \tag{2.6}$$

The linear system (2.6) is fully determined for any distinct evaluation domain $\{x_i\}_{i\in[n+1]}$ such that $x_i \neq 0 \, \forall i$. □

Remark 2.1.2. *Laurent polynomials are undefined at $x = 0$.*

2.3 Real-valued Discrete Fourier Transform

RDFT is the DFT that maps Real-valued evaluation sequences. Formally, it is the evaluation over the domain $\{\omega^\ell\}_{\ell\in[n]}$ of the conjugate-symmetric polynomial with Complex coefficients

$$L(x) = c_0 + \sum_{i=1}^{\frac{n}{2}} c_i x^i + c_i^* x^{-i} \tag{2.7}$$

where ω is an n'th root-of-unity for an even integer n and c_0 and $c_{\frac{n}{2}}$ are Real, (see Section 2.4 of [Dom23]). The evaluation at the ℓ'th power of ω results in

$$L(\omega^\ell) = c_0 + \sum_{i=1}^{\frac{n}{2}} c_i \omega^{\ell i} + c_i^* \omega^{-\ell i} \tag{2.8}$$

$$= c_0 + 2(-1)^\ell c_{\frac{n}{2}} + \sum_{i=1}^{\frac{n}{2}-1} c_i \omega^{\ell i} + c_i^* \omega^{-\ell i} \tag{2.9}$$

$$= r_0 + 2(-1)^\ell r_{\frac{n}{2}} + 2\sum_{i=1}^{\frac{n}{2}-1} r_i \cos(i\ell\theta) - q_i \sin(i\ell\theta) \tag{2.10}$$

where $c_i = r_i + \mathbf{i}q_i$, with r_i and q_i being the Real and Imaginary components of c_i, respectively. The conjugate of c_i is given by $c_i^* = r_i - \mathbf{i}q_i$.

Evidently, the resulting translation $\{r_0, c_1, ..., c_{\frac{n}{2}-1}, r_{\frac{n}{2}}\} \rightarrow \{L(\omega^\ell)\}_{\ell\in[n]}$ transforms conjugate-symmetric coefficient vectors to their Real-valued evaluation sequences. The RDFT can be presented in matrix form as

$$\begin{bmatrix} L(1) \\ L(\omega) \\ L(\omega^2) \\ \vdots \\ L(\omega^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega^1 & \omega^2 & ... & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & ... & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & ... & \omega^{(n-1)^2} \end{bmatrix} \begin{bmatrix} r_0 \\ \mathbf{c}_n \\ 2r_{\frac{n}{2}} \\ \mathbf{c}_n^\dagger \end{bmatrix} \tag{2.11}$$

where $\mathbf{c}_n \equiv \left[c_1, ..., c_{\frac{n}{2}-1}\right]$ and $\mathbf{c}_n^\dagger \equiv \left[c_{\frac{n}{2}-1}^*, ..., c_1^*\right]$, its flipped-conjugate version. The linear system in (2.11) is a DFT matrix, (see Chapter 1 of [Dom23]). As such, it is invertible, satisfies Parseval's identity, and facilitates efficient computation.

The Laurent version of (2.11) can be expressed as

$$\begin{bmatrix} L(1) \\ L(\omega) \\ L(\omega^2) \\ \vdots \\ L(\omega^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & ... & 1 & 1 & 1 & ... & 1 \\ \omega^{-\frac{n}{2}} & ... & \omega^{-1} & 1 & \omega & ... & \omega^{\frac{n}{2}} \\ \omega^{-n} & ... & \omega^{-2} & 1 & \omega^2 & ... & \omega^{n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega^{-\frac{n}{2}(n-1)} & ... & \omega^{-(n-1)} & 1 & \omega^{n-1} & ... & \omega^{\frac{n}{2}(n-1)} \end{bmatrix} \begin{bmatrix} r_{\frac{n}{2}} \\ \mathbf{c}_n^\dagger \\ r_0 \\ \mathbf{c}_n \\ r_{\frac{n}{2}} \end{bmatrix} \tag{2.12}$$

$$= \begin{bmatrix} 1 & ... & 1 & 1 & 1 & ... & 1 \\ \omega^{-\frac{n}{2}} & ... & \omega^{-1} & 1 & \omega & ... & \omega^{\frac{n}{2}-1} \\ \omega^{-n} & ... & \omega^{-2} & 1 & \omega^2 & ... & \omega^{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega^{-\frac{n}{2}(n-1)} & ... & \omega^{-(n-1)} & 1 & \omega^{n-1} & ... & \omega^{(\frac{n}{2}-1)(n-1)} \end{bmatrix} \begin{bmatrix} 2r_{\frac{n}{2}} \\ \mathbf{c}_n^\dagger \\ r_0 \\ \mathbf{c}_n \end{bmatrix} \tag{2.13}$$

$$= \begin{bmatrix} 1 & & & & \\ & \omega^{-\frac{n}{2}} & & & \\ & & \omega^{-n} & & \\ & & & \ddots & \\ & & & & \omega^{-\frac{n}{2}(n-1)} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega^1 & \omega^2 & ... & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & ... & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & ... & \omega^{(n-1)^2} \end{bmatrix} \begin{bmatrix} 2r_{\frac{n}{2}} \\ \mathbf{c}_n^\dagger \\ r_0 \\ \mathbf{c}_n \end{bmatrix} \tag{2.14}$$

where (2.12) follows by definition of (2.8), in (2.13) we use the fact that the right and left columns of the transform matrix are identical, and in (2.14) we extract the diagonal shift matrix which yields the standard DFT matrix from (2.11).

3 Constructing RC Codes

In this section, we will demonstrate that an RC code is a linear subset of an RS code. As such, it maintains a minimum distance that is at least equal to that of the RS code while featuring a reduced number of nearest neighbors. These properties ensure that the performance of RC codes in STARKs is at least as good as that of the original RS code, with potential improvement.

3.1 A Complex-valued RS(n,k+1) Code

Consider a Complex field $\mathbb{C}_\mathbb{F}$ that has an n'th root-of-unity ω where n is an even integer, and let a codeword polynomial over $\mathbb{C}_\mathbb{F}$ be the evaluations sequence on domain $\{w^\ell\}_{\ell\in[n]}$ of the following polynomial

$$L(x) = \sum_{i=-\frac{k}{2}}^{\frac{k}{2}} c_i x^i \tag{3.1}$$

where $k < n$ is an even integer. Then the code $\mathcal{C}$ is the construction that sends any coefficient $(k+1)$-tuple $\{c_{-\frac{k}{2}}, ..., c_{\frac{k}{2}}\}$ to a length-n evaluations sequence of its corresponding polynomial. Namely,

$$\mathcal{C} : \{c_{-\frac{k}{2}}, ..., c_{\frac{k}{2}}\} \to \{L(1), L(\omega), L(\omega^2), ..., L(\omega^{n-1})\} \tag{3.2}$$

Theorem 3.1. *The code $\mathcal{C}$ is $RS(n, k+1)$.*

Proof. By Corollary 2.1.2, the coefficients $\{c_{-\frac{k}{2}}, ..., c_{\frac{k}{2}}\}$ of a codeword polynomial of $\mathcal{C}$ can be recovered using any length-$(k+1)$ sub-sequence of its corresponding evaluations sequence. Then by Corollary 2.1.1 it is $RS(n, k+1)$ since this defines a system with $\binom{n}{k+1}$ determinations. □

3.2 A Real-valued RS(n,k+1) Code

Consider a linear subcode of $\mathcal{C}$ defined by the conjugate-symmetric polynomials over $\mathbb{C}_{\mathbb{F}}$

$$L'(x) = r_0 + \sum_{i=1}^{\frac{k}{2}} c_i x^i + c_i^* x^{-i} \tag{3.3}$$

where the free coefficient r_0 is Real. In a similar manner to (2.10), one can show that the evaluations

$$L'(\omega^\ell) = r_0 + 2\sum_{i=1}^{\frac{k}{2}} r_i \cos(i\ell\theta) - q_i \sin(i\ell\theta) \tag{3.4}$$

are Real-valued for all $\{\omega^\ell\}_{\ell\in[n]}$. Then the code $\mathcal{C}'$ is the construction that sends any coefficient $\left(\frac{k}{2}+1\right)$-tuple $\{r_0, c_1, ..., c_{\frac{k}{2}}\}$ where r_0 is Real, to a length-n, Real-valued evaluations sequence of its corresponding polynomial. Namely,

$$\mathcal{C}' : \{r_0, c_1, ..., c_{\frac{k}{2}}\} \to \{L'(1), L'(\omega), L'(\omega^2), ..., L'(\omega^{n-1})\} \tag{3.5}$$

Theorem 3.2. *The code $\mathcal{C}'$ is $RS(n, k+1)$.*

Proof. Take any length-$k+1$ sub-sequence of a codeword of $\mathcal{C}'$. Then, by definition of the codeword polynomial (3.3), the following linear relation holds

$$\begin{bmatrix} L'(\alpha_0) \\ L'(\alpha_1) \\ \vdots \\ L'(\alpha_k) \end{bmatrix} = \begin{bmatrix} \alpha_0^{-\frac{k}{2}} & ... & \alpha_0^{-1} & 1 & \alpha_0 & ... & \alpha_0^{\frac{k}{2}} \\ \alpha_1^{-\frac{k}{2}} & ... & \alpha_1^{-1} & 1 & \alpha_1 & ... & \alpha_1^{\frac{k}{2}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_k^{-\frac{k}{2}} & ... & \alpha_k^{-1} & 1 & \alpha_k & ... & \alpha_k^{\frac{k}{2}} \end{bmatrix} \begin{bmatrix} c_{\frac{k}{2}}^* \\ \vdots \\ c_1^* \\ r_0 \\ c_1 \\ \vdots \\ c_{\frac{k}{2}} \end{bmatrix} \tag{3.6}$$

$$= \begin{bmatrix} \alpha_0^{-\frac{k}{2}} & & \\ & \ddots & \\ & & \alpha_k^{-\frac{k}{2}} \end{bmatrix} \begin{bmatrix} 1 & \alpha_0 & ... & \alpha_0^k \\ 1 & \alpha_1 & ... & \alpha_1^k \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_k & ... & \alpha_k^k \end{bmatrix} \begin{bmatrix} c_{\frac{k}{2}}^* \\ \vdots \\ c_1^* \\ r_0 \\ c_1 \\ \vdots \\ c_{\frac{k}{2}} \end{bmatrix} \tag{3.7}$$

where $\alpha_i \in \{1, \omega, \omega^2, ..., \omega^{n-1}\}$, $\alpha_i \neq \alpha_j \; \forall i \neq j$. Since the r.h.s. matrix in (3.7) is square Vandermonde, the linear system has $\binom{n}{k+1}$ determinations $(\alpha_0, \alpha_1, ..., \alpha_k)$ and by Corollary 2.1.1, it follows that $\mathcal{C}$ is $RS(n, k+1)$. □

Remark 3.2.1. *For both $\mathcal{C}$ and $\mathcal{C}'$, we think of codeword symbols as Complex. As such, the definition of Hamming distance is the number of Complex symbols that differ between two codewords.*

Remark 3.2.2. *The code $\mathcal{C}'$ is the Real projection of $\mathcal{C}$.*

Theorem 3.3. *The code $\mathcal{C}$ is the Cartesian product of $\mathcal{C}'$ with itself.*

Proof. Let us start by counting the number of codewords in each code. Code $\mathcal{C}$ has $k+1$ Complex coefficients. Counting Real degrees-of-freedom (dof) this is $2k+2$ dof. Code $\mathcal{C}'$ has $\frac{k}{2}$ Complex and one Real coefficients which amounts to $k+1$ dof. This already shows that the code sizes relate as $|\mathcal{C}| = |\mathcal{C}'|^2$.

Now, take any polynomial coefficients vector $\mathbf{c} = \left[c_{-\frac{k}{2}}, ..., c_{\frac{k}{2}}\right]$ of $\mathcal{C}$ and compute the following two sums

$$\begin{aligned} \mathbf{c}'_0 &= \frac{\mathbf{c} + \mathbf{c}^\dagger}{2} \\ \mathbf{c}'_1 &= \frac{\mathbf{c} - \mathbf{c}^\dagger}{2\mathbf{i}} \end{aligned} \tag{3.8}$$

It is simple to verify that $\mathbf{c}'_0$ and $\mathbf{c}'_1$ are orthogonal, that they are both conjugate-symmetric and that $\mathbf{c} = \mathbf{c}'_0 + i\mathbf{c}'_1$. Thus, both $\mathbf{c}'_0$ and $\mathbf{c}'_1$ are valid coefficient vectors of $\mathcal{C}'$. □

3.3 RC Codes

To construct the RC code, let us take $\mathcal{C}'$ and expurgate[1] it by setting $q_{\frac{k}{2}}$, the imaginary part of the coefficient $c_{\frac{k}{2}}$ to zero. The resulting code is a length-n Real-valued (n, k) code. Namely,

$$\mathcal{C}'' : \{r_0, c_1, ..., c_{\frac{k}{2}-1}, r_{\frac{k}{2}}\} \to \{L''(1), L''(\omega), L''(\omega^2), ..., L''(\omega^{n-1})\} \tag{3.9}$$

where

$$L''(x) = r_0 + r_{\frac{k}{2}}(x^{\frac{k}{2}} + x^{-\frac{k}{2}}) + \sum_{i=1}^{\frac{k}{2}-1} c_i x^i + c_i^* x^{-i} \tag{3.10}$$

Theorem 3.4. *Define $\mathcal{B}_{MDS}$ as the hypersphere of radius $n-k+1$, then the code $\mathcal{C}''$ is an almost-MDS, linear (n, k) code with most of its nearest neighbors on the spherical shell of $\mathcal{B}_{MDS}$ except for at most a $\frac{1}{n-k}$ fraction of them at distance $d_{min} = n-k$.*

[1] Code expurgation is the process of taking a reference code and forming a new code whose codebook is a subset of the reference code's codebook.

Proof. Since $RC(n,k)$ is an expurgation of $RS(n,k+1)$, its minimal distance is $d_{\min} = n-k$. To analyze other distances, let us try to determine the coefficients from a length-k subsequence of a codeword of $\mathcal{C}''$ using the following linear system

$$\begin{bmatrix} L''(\alpha_0) \\ L''(\alpha_1) \\ \vdots \\ L''(\alpha_{k-1}) \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha_0^{-\frac{k}{2}} & & & \\ & \ddots & & \\ & & \alpha_{k-1}^{-\frac{k}{2}} & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & \alpha_0 & ... & \alpha_0^{k-1} & \alpha_0^k \\ 1 & \alpha_1 & ... & \alpha_1^{k-1} & \alpha_1^k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{k-1} & ... & \alpha_{k-1}^{k-1} & \alpha_{k-1}^k \\ -1 & 0 & ... & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \\ r_0 \\ \mathbf{c}_k \\ r_{\frac{k}{2}} \end{bmatrix} \tag{3.11}$$

where the bottom-row of the system is an auxiliary equation enforcing that $r_{\frac{k}{2}} = r_{\frac{k}{2}}$. A fully-determined system is equivalent to non-singularity of the r.h.s. matrix in (3.11). Using the Laplace Expansion it is possible to compute the determinant of that matrix as

$$\left| \begin{bmatrix} 1 & \alpha_0 & ... & \alpha_0^{k-1} & \alpha_0^k \\ 1 & \alpha_1 & ... & \alpha_1^{k-1} & \alpha_1^k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{k-1} & ... & \alpha_{k-1}^{k-1} & \alpha_{k-1}^k \\ -1 & 0 & ... & 0 & 1 \end{bmatrix} \right| = \left(1 - \prod_{i=0}^{k-1} \alpha_i\right) \left| \begin{bmatrix} 1 & \alpha_0 & ... & \alpha_0^{k-1} \\ 1 & \alpha_1 & ... & \alpha_1^{k-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{k-1} & ... & \alpha_{k-1}^{k-1} \end{bmatrix} \right| \tag{3.12}$$

where the determinant is zero only if

$$\prod_{i=0}^{k-1} \alpha_i = 1 \tag{3.13}$$

$$\Rightarrow \prod_{i=0}^{k-1} \omega^{\beta_i} = 1 \tag{3.14}$$

$$\Rightarrow \sum_{i=0}^{k-1} \beta_i = 0 \mod n \tag{3.15}$$

where $\beta_i \in [n]$, $\beta_i \neq \beta_j \; \forall i \neq j$. The above sum can be rewritten as $\beta_0 + \sum_{i=1}^{k-1} \beta_i = 0 \mod n$ where by elementary considerations, regardless of the value of $\sum_{i=1}^{k-1} \beta_i$, there is at most a single choice for β_0 that leads to a solution and thus at least $n-k-1$ that won't. This means that at least $\frac{n-k-1}{n-k}\binom{n}{k}$ of the determinations of (3.11) lead to non-singular systems and a distance of $d = n-k+1$. □

4 Working with RC Codes

This section presents key algorithms for working with RC codes, focusing on LDE and various polynomial operations. These techniques are essential for efficiently handling and manipulating data within a STARK framework.

4.1 Low Degree Extension

LDE is the process of interpolating data by treating the input as evaluations of a low-degree polynomial over a small domain and computing the polynomial's evaluations over a larger

domain. The evaluations on the large domain are defined as before

$$\{L(1), L(\omega), L(\omega^2), ..., L(\omega^{n-1})\} \tag{4.1}$$

and on the small domain as (4.1)'s subset

$$\{L(1), L(\mu), L(\mu^2), ..., L(\mu^{k-1})\} \tag{4.2}$$

where μ is the k'th root-of-unity such that $\mu = \omega^{\frac{n}{k}}$. The coefficients corresponding to (4.2) can be recovered using (2.11) as

$$\begin{bmatrix} r_0 \\ \mathbf{c}_k \\ 2r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \end{bmatrix} = F_\mu^{-1} \begin{bmatrix} L(1) \\ L(\mu) \\ L(\mu^2) \\ \vdots \\ L(\mu^{k-1}) \end{bmatrix} \tag{4.3}$$

where F_μ^{-1} is the Inverse DFT matrix of order k. Prior to evaluating over the larger domain, the coefficients vector must be partitioned as follows

$$\begin{bmatrix} r_0 \\ \mathbf{c}_k \\ r_{\frac{k}{2}} \\ 0 \\ \vdots \\ 0 \\ r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \end{bmatrix} = \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & \frac{1}{2} & & & & \\ & & & 0 & & & & \\ & & & \vdots & & & & \\ & & & 0 & & & & \\ & & & \frac{1}{2} & & & & \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \end{bmatrix} \begin{bmatrix} r_0 \\ \mathbf{c}_k \\ 2r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \end{bmatrix} \tag{4.4}$$

Finally, we can evaluate over the large domain as follows

$$\begin{bmatrix} L(1) \\ L(\omega) \\ L(\omega^2) \\ \vdots \\ L(\omega^{n-1}) \end{bmatrix} = F_\omega \begin{bmatrix} r_0 \\ \mathbf{c}_k \\ r_{\frac{k}{2}} \\ 0 \\ \vdots \\ 0 \\ r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \end{bmatrix} \tag{4.5}$$

where F_ω is the DFT matrix of order n.

The overall process involves performing an Inverse RDFT of order k, followed by an RDFT of order n. A more memory-efficient approach computes evaluations over cosets

of the large domain, rather than evaluating over the entire domain at once. The primary advantage of this method is that a single large DFT is replaced by multiple smaller ones. Additionally, the zero'th coset does not require re-evaluation, as it corresponds to the original data. To demonstrate this approach, let us evaluate over the i-th coset $\{\omega^i, \omega^i\mu, \omega^i\mu^2, ..., \omega^i\mu^{k-1}\}$. Plugging the evaluation coset into (3.6) leads to

$$\begin{bmatrix} L(\omega^i) \\ L(\omega^i\mu) \\ \vdots \\ L(\omega^i\mu^{k-1}) \end{bmatrix} = \begin{bmatrix} \omega^{-\frac{k}{2}i} & ... & 1 & ... & \omega^{\frac{k}{2}i} \\ \omega^{-\frac{k}{2}i}\mu^{-\frac{k}{2}} & ... & 1 & ... & \omega^{\frac{k}{2}i}\mu^{\frac{k}{2}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega^{-\frac{k}{2}i}\mu^{-\frac{k}{2}(k-1)} & ... & 1 & ... & \omega^{\frac{k}{2}i}\mu^{\frac{k}{2}(k-1)} \end{bmatrix} \begin{bmatrix} r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \\ r_0 \\ \mathbf{c}_k \\ r_{\frac{k}{2}} \end{bmatrix} \tag{4.6}$$

$$= \begin{bmatrix} 1 & ... & 1 \\ \mu^{-\frac{k}{2}} & ... & \mu^{\frac{k}{2}} \\ \vdots & \vdots & \vdots \\ \mu^{-\frac{k}{2}(k-1)} & ... & \mu^{\frac{k}{2}(k-1)} \end{bmatrix} \begin{bmatrix} \omega^{-\frac{k}{2}i} & & & \\ & \omega^{(-\frac{k}{2}+1)i} & & \\ & & \ddots & \\ & & & \omega^{\frac{k}{2}i} \end{bmatrix} \begin{bmatrix} r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \\ r_0 \\ \mathbf{c}_k \\ r_{\frac{k}{2}} \end{bmatrix} \tag{4.7}$$

$$= \begin{bmatrix} 1 & ... & 1 \\ \mu^{(-\frac{k}{2}+1)} & ... & \mu^{\frac{k}{2}} \\ \vdots & \vdots & \vdots \\ \mu^{(-\frac{k}{2}+1)(k-1)} & ... & \mu^{\frac{k}{2}(k-1)} \end{bmatrix} \begin{bmatrix} \omega^{(-\frac{k}{2}+1)i} & & & \\ & \ddots & & \\ & & \omega^{(\frac{k}{2}-1)i} & \\ & & & \mathrm{Re}\left(\omega^{\frac{k}{2}i}\right) \end{bmatrix} \begin{bmatrix} \mathbf{c}_k^\dagger \\ r_0 \\ \mathbf{c}_k \\ 2r_{\frac{k}{2}} \end{bmatrix} \tag{4.8}$$

$$= F_\mu \Lambda_i \begin{bmatrix} r_0 \\ \mathbf{c}_k \\ 2r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \end{bmatrix} \tag{4.9}$$

$$\tag{4.10}$$

where in (4.7) we partition the coset modulation (i.e. multiplication by powers of ω^i) and represent it as right-multiplication by a diagonal matrix. In (4.8) we combine the left and right columns of the transform matrix since they are equal. Finally, in (4.9) we rotate the coefficients vector counterclockwise $\frac{k}{2} - 1$ times resulting in standard coset interpolation, where F_μ is the DFT matrix of order k and

$$\Lambda_i \equiv \mathrm{diag}\left(1, \omega^i, ..., \omega^{(\frac{k}{2}-1)i}, \mathrm{Re}\left(\omega^{\frac{k}{2}i}\right), \omega^{(\frac{k}{2}+1)i}, ..., \omega^{(k-1)i}\right)$$

The overall complexity of evaluating over a coset amounts to a single DFT and some linear work.

4.2 Polynomial Sums and Products

As we shall see in Section 5, polynomial sums and products play a crucial role in defining STARK constraints. A typical constraint polynomial is a ratio of two polynomials [Tea23], where the numerator is a functional composition over an RC codeword, comprising sums and products. The following theorem demonstrates that this functional composition results in an RC codeword, potentially of a higher rate.

Theorem 4.1. *The sum of two RC(n,k) codewords is an RC(n,k) codeword. The product of two RC(n,k) codewords is an RC(n,2k) codeword given that $n \geq 2k$.*

Proof. Denote by $\mathbf{c}''$ the coefficients vector of an RC codeword

$$\mathbf{c}'' \equiv \begin{bmatrix} r_{\frac{k}{2}} & \mathbf{c}_k^\dagger & r_0 & \mathbf{c}_k & r_{\frac{k}{2}} \end{bmatrix} \tag{4.11}$$

Then sum case is straightforward: adding two codewords with coefficient vectors of the form (4.11) results in a coefficients vector that retains the same symmetric form, ensuring the result is an $\mathrm{RC}(n, k)$ codeword.

For the product of two codewords, we rely on the property that polynomial multiplication is equivalent to the convolution of their respective coefficient vectors (see Section 1.3 of [Dom23]). Convolving two vectors in the Laurent form (4.11) yields a conjugate-symmetric vector with $2k+1$ coefficients, where the left, middle, and right coefficients are Real. As a result, the corresponding coefficients vector forms an $RC(n, 2k)$ codeword in the Laurent form. When $n = 2k$, the length-$(2k + 1)$ coefficients vector folds onto itself, causing the leftmost and rightmost coefficients to alias into a single coefficient of value $2r_{\frac{n}{2}}$. □

For consistency, we define the code $RC(n, n)$ as the self-map corresponding to (2.11). It is important to note that multiplying two $RC(n, n)$ codewords cannot be done using this method due to the aliasing effect.

4.3 Polynomial Quotients

The denominator of a STARK constraint is a polynomial designed to vanish on a specific subdomain where the numerator zeroes. In other words, the denominator introduces poles only where the numerator has zeros, ensuring the resulting quotient is a polynomial. Theorem (4.1) guarantees that the numerator is an RC codeword and, consequently, Real. However, a simple pole of the form $x - \omega^\ell$ has non-Real evaluations. In this section, we present ways to ensures that the evaluation of the denominator polynomial remains Real.

Lemma 4.2. *The conjugate-symmetric polynomial*

$$x^{-1} - 2Re(\omega^\ell) + x \tag{4.12}$$

vanishes only for $x = \omega^{\pm\ell}$.

Proof. The polynomial (4.12) is an order 2 Laurent polynomial and thus has two roots. By examination it can easily be verified that $x = \omega^{\pm\ell}$ are the roots of (4.12). □

Lemma 4.3. *The conjugate-symmetric polynomial*

$$\mathbf{i}x^{-\frac{n}{2}} - \mathbf{i}x^{\frac{n}{2}} \tag{4.13}$$

vanishes only for $x \in \{\omega^\ell\}_{\ell\in[n]}$.

Proof. The polynomial (4.13) is an order n Laurent polynomial and thus has n roots. Since it is undefined for $x = 0$, it can be expressed as

$$-\mathbf{i}x^{-\frac{n}{2}}(x^n - 1) = -\mathbf{i}x^{-\frac{n}{2}} \prod_{\ell=0}^{n-1} (x - \omega^\ell) \tag{4.14}$$

exposing the desired roots. □

Lemma 4.4. *Let $L(x)$ be a Laurent polynomial of order n, and denote by $\mathcal{R}_L$ the set of all of its roots (with multiplicities) from $\{\omega^\ell\}_{\ell\in[n]}$. Then $L(x^{-1})$ is also a Laurent polynomial of order n, and the set of conjugates of all elements in $\mathcal{R}_L$ (with multiplicities) consists of all of its roots from $\{\omega^\ell\}_{\ell\in[n]}$.*

Proof. Denote $\widetilde{L}(x) \equiv L(x^{-1})$, then if $L(\omega^\ell) = 0$ then $\widetilde{L}((\omega^\ell)^*) = \widetilde{L}(\omega^{-\ell}) = L(\omega^\ell) = 0$ □

Theorem 4.5. *Let $L(x)$ be a Laurent polynomial of order n and $\alpha_i \in \{\omega^\ell\}_{\ell\in[n]}\ \forall i$, then the rational function*

$$\frac{L(x)L(x^{-1})}{\prod_i (x^{-1} - 2Re(\alpha_i) + x)} \tag{4.15}$$

is a polynomial of order at most $2n$ if

$$\frac{L(x)}{\prod_i (x - \alpha_i)} \tag{4.16}$$

is a polynomial of order at most n.

Proof. The theorem follows immediately by integrating Lemmas 4.4 and 4.2. □

5 RC Codes and STARKs

In this section, we explore two central aspects in the interplay between RC codes and STARKs. First, we examine the algebraic constraints imposed by STARKs and outline strategies for efficiently computing their LDEs. Next, we introduce methods for proximity testing (i.e. FRI) of RC codes within the STARK framework.

5.1 AIR Constraints

Algebraic Intermediate Representation (AIR) constraints in STARKs encompass several types [Tea23]. In this subsection, we aim to present a general strategy for efficiently computing LDEs for these constraints. References to polynomials often pertain to their LDEs.
To clarify the concept, we introduce the following definitions:

1. Define the field tower $\mathbb{F} \subset \mathbb{C}_\mathbb{F} \subset \mathbb{C}_\mathbb{F}^2$ where each successive field is a quadratic extension of the previous one.

2. Let ω be the n'th root-of-unity in $\mathbb{C}_\mathbb{F}$ and denote by $\mathcal{R}_\omega$ the cyclic group generated by it, $\{\omega^\ell\}_{\ell\in[n]}$.

3. Define $\mathbb{C}_\mathbb{F}[x, x^{-1}; n]$ and $\mathbb{C}_\mathbb{F}^2[x, x^{-1}; n]$ as the sets[2] of Laurent polynomials of order at most n, with coefficients in their respective fields. Note that $\mathbb{C}_\mathbb{F}[x, x^{-1}; n] \subset \mathbb{C}_\mathbb{F}^2[x, x^{-1}; n]$.

[2] Define sets rather than rings to avoid dealing with products whose order exceeds n.

Simple AIR constraints involve expressions like

$$\frac{f(t_0(x), t_1(x), ...)}{v(x)} \tag{5.1}$$

where $f(x)$ is a composition involving only sums and products, $t_i(x)$ are polynomials with evaluations in $\mathbb{F}$ over the domain $\mathcal{R}_\omega$, and $v(x)$ is a polynomial that vanishes over a subset of $\mathcal{R}_\omega$. By Theorem 4.1, defining the trace polynomials $t_i(x)$ using RC codes leads to the nominator of (5.1) having evaluations in $\mathbb{F}$ (i.e. it is an RC codeword). To keep the quotient (5.1) having evaluations in $\mathbb{F}$ requires using Lemmas 4.2, 4.3, or Theorem 4.5. Alternatively, the quotient is well-defined in $\mathbb{C}_\mathbb{F}[x, x^{-1}; n]$, but potentially has evaluations in $\mathbb{C}_\mathbb{F}$ (i.e. it is not an RC codeword).

A more complicated constraint, such as Out Of Domain (OOD) sampling [Tea23] involves expressions like

$$\frac{c(x) - c(z)}{x - z} \tag{5.2}$$

where $c(x)$ is the Composition Polynomial (CP) and $z \in \mathbb{C}^2_\mathbb{F}$. In these cases, the resulting polynomial is well-defined in $\mathbb{C}^2_\mathbb{F}[x, x^{-1}; n]$, with evaluations in $\mathbb{C}^2_\mathbb{F}$.
Similarly, a Randomized AIR with Preprocessing (RAP) constraint [Gab24] involves expressions like

$$\frac{t_0(x) + zt_1(x)}{v(x)} \tag{5.3}$$

where $z \in \mathbb{C}^2_\mathbb{F}$ and $t_i(x)$ and $v(x)$ are as previously defined. In these cases, the resulting polynomial can be treated as a tuple of polynomials in $\mathbb{C}_\mathbb{F}[x, x^{-1}; n]$ with evaluations in either $\mathbb{F}$ or $\mathbb{C}_\mathbb{F}$ depending on the denominator, as in (5.1).

Remark 5.0.1. *Unlike the DCCT, the DFT polynomials in this context are well-defined over their respective fields, with the exception at $x = 0$.*

5.2 FRI

A Fast RS Interactive Oracle Proof of Proximity (FRI) [BSBHR18a] is an interactive protocol that probabilistically verifies whether a given sequence is an RS codeword of a specified rate. The protocol's soundness hinges primarily on the code's minimum distance and the number of nearest neighbors. Although RC codes are expurgated RS codes with enhanced distance properties, as discussed in Section 3, we conservatively bound our soundness by that of the original RS code. Additionally, while an honest Prover is expected to use RC codewords, a dishonest one may use RS codewords, potentially evading detection.

Assume that k is a power-of-two, and we are given an evaluations sequence of length-n corresponding to a Laurent polynomial of order k. Our goal is to test whether it is an $RC(n, k)$ codeword. However, rather than directly verifying this, we opt to test whether it is a Complex $RS(n, k+1)$ codeword, which amounts to performing a low-degree test on the associated polynomial.

Remark 5.0.2. *In a typical FRI, k is a power-of-two, and the tested codeword belongs to an $RS(n, k)$ code. In our case, the codeword belongs to an $RS(n, k+1)$ code, resulting in a slight soundness degradation, as the minimum distance increases from $n-k+1$ to $n-k$.*

By definition, an $RC(n,k)$ codeword (in the honest case) is an LDE of the form:

$$\begin{bmatrix} L(1) \\ L(\omega) \\ L(\omega^2) \\ \vdots \\ L(\omega^{n-1}) \end{bmatrix} = F_\omega \begin{bmatrix} r_0 \\ \mathbf{c}_k \\ r_{\frac{k}{2}} \\ 0 \\ \vdots \\ 0 \\ r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \end{bmatrix} \tag{5.4}$$

where F_ω is a DFT matrix of order n. Rotating the coefficients vector clockwise by $\frac{k}{2}$ positions corresponds to multiplying the r.h.s. of (5.4) by the diagonal shift matrix $\Lambda \equiv \text{diag}(1, \omega^{-\frac{k}{2}}, \omega^{-2\frac{k}{2}}, \ldots, \omega^{-(n-1)\frac{k}{2}})$. This results in the FRI canonical form for conjugate-symmetric Laurent polynomials

$$\Lambda^* \begin{bmatrix} L(1) \\ L(\omega) \\ L(\omega^2) \\ \vdots \\ L(\omega^{n-1}) \end{bmatrix} = F_\omega \begin{bmatrix} r_{\frac{k}{2}} \\ \mathbf{c}_k^\dagger \\ r_0 \\ \mathbf{c}_k \\ r_{\frac{k}{2}} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.5}$$

As noted earlier, we make no additional assumptions about the coefficients vector, other than the fact that it represents a polynomial of degree at most $k+1$. This means the forthcoming FRI protocol verifies a coefficients vector of the form:

$$\begin{bmatrix} \widetilde{c}_0 & \ldots & \widetilde{c}_{k-1} & \widetilde{c}_k & 0 & \ldots & 0 \end{bmatrix} \tag{5.6}$$

instead of the one in (5.5). The coefficients $\widetilde{c}_0, \ldots, \widetilde{c}_{k-1}$ are assumed to be Complex, with the possible exception of $\widetilde{c}_k$, which can be verified to be Real.

We now outline two methods to implement FRI in this setting. The first method reduces the problem to a standard FRI, while the second introduces a slightly modified FRI tailored to our case, though we provide it with no soundness analysis.

5.2.1 Method 1

The polynomial of degree k corresponding to (5.6) is given by

$$L(x) = \sum_{i=0}^{k} \widetilde{c}_i x^i \tag{5.7}$$

To reduce it to a polynomial of degree $k-1$ we compute

$$\widetilde{L}(x) = \frac{L(x) - \widetilde{c}_0}{x} = \sum_{i=0}^{k-1} \widetilde{c}_{i+1} x^i \tag{5.8}$$

The core idea in this method is to transmit $\widetilde{c}_0$ as cleartext and perform a standard FRI on $\widetilde{L}(x)$.

5.2.2 Method 2

Performing $\log k$ rounds of FRI (i.e. two-to-one folds) on a length-n evaluations sequence corresponding to a polynomial of the form (5.7) results in $\frac{n}{k}$ evaluations of the following linear polynomial

$$(\rho_0\widetilde{c}_0 + \rho_1\widetilde{c}_1 + ... + \rho_{k-1}\widetilde{c}_{k-1}) + \rho_0\widetilde{c}_k x \tag{5.9}$$

where the ρ_i's represent the random multipliers applied during the protocol. The low-degree test is then equivalent to verifying that the resulting evaluations correspond to a linear polynomial with a slope of $\rho_0\widetilde{c}_k$.

6 Implementation Aspects

This section explores key implementation aspects of RDFT, which is integral to the efficient processing of RC codes. We examine fast RDFT algorithms and provide a concrete complexity analysis, highlighting optimizations for specific cases such as for the M31 finite field.

6.1 Faster Fourier Transform for RDFT

An RDFT can be expressed as a DFT linear system F

$$\mathbf{C} = F\mathbf{c} \tag{6.1}$$

where the evaluations vector $\mathbf{C}$ is Real-valued and the coefficients vector $\mathbf{c}$ obeys the conjugate-symmetry in (2.11). Since RDFT is a DFT for a subset of the Complex domain, all FFT methods that are applicable to DFTs are applicable here. Additionally, since the resulting evaluations vectors are guaranteed to be Real-valued, we can process two independent coefficient vectors $\mathbf{c}_0$ and $\mathbf{c}_1$ concurrently by transforming their orthogonal sum $\mathbf{c}_0 + \mathbf{i}\mathbf{c}_1$ as

$$F(\mathbf{c}_0 + \mathbf{i}\mathbf{c}_1) = F\mathbf{c}_0 + \mathbf{i}F\mathbf{c}_1 = \mathbf{C}_0 + \mathbf{i}\mathbf{C}_1 \tag{6.2}$$

where the Real and Imaginary parts of the resulting transform correspond to the two independent Real-valued transforms $\mathbf{C}_0$ and $\mathbf{C}_1$. The overall complexity of this scheme is bounded by the minimal complexity of a length-n FFT of $\mathcal{O}(n \log n)$.

The Inverse RDFT can be computed in much the same way as

$$\mathbf{c}_0 + \mathbf{i}\mathbf{c}_1 = F^{-1}(\mathbf{C}_0 + \mathbf{i}\mathbf{C}_1) \tag{6.3}$$

where the conjugate-symmetry of the coefficient vectors is used to partition the two inverse transforms. Denoting by $\mathbf{c} = \mathbf{c}_0 + \mathbf{i}\mathbf{c}_1$ the resulting inverse transform of (6.3), the conjugate-symmetry property implies that

$$\mathbf{c}^\dagger = \mathbf{c}_0^\dagger - \mathbf{i}\mathbf{c}_1^\dagger = \mathbf{c}_0 - \mathbf{i}\mathbf{c}_1 \tag{6.4}$$

The independent coefficient vectors can thus be obtained by

$$\begin{aligned}\mathbf{c}_0 &= \frac{\mathbf{c} + \mathbf{c}^\dagger}{2} \\ \mathbf{c}_1 &= \frac{\mathbf{c} - \mathbf{c}^\dagger}{2\mathbf{i}}\end{aligned} \tag{6.5}$$

Note that direct decimation-decomposition [Bri88] can provide slightly better constant complexity than the suggested method, but for sufficiently-large FFTs, the constant improvement diminishes.

6.2 Concrete Complexity Analysis

The number of required multiplications for a radix M FFT is given by

$$\mathcal{N}_{\text{FFT}} = \frac{N \log N}{M \log M} \cdot (M - 1 + \text{rdx}(M)) \tag{6.6}$$

where $\frac{N \log N}{M \log M}$ represents the number of radix blocks, $M-1$ is the number of twiddle factor multiplications per block, and $\text{rdx}(M)$ refers to the number of multiplications needed to compute a single radix block. For radix-2, this expression simplifies to $\frac{N}{2} \log_2 N$.

In this analysis, we assume that a Complex multiplication costs three Real multiplications by applying the Karatsuba algorithm [İC17]. Furthermore, since a Complex FFT can compute two Real-valued transforms simultaneously, as shown in Section 6.1, the cost of a single Real-valued Complex FFT can be amortized. Overall, combining these two optimizations, results in an RDFT that is $\frac{3}{2}$ times more computationally expensive than a DCCT. Table 1 presents a comparison of the number of required multiplications for different scenarios. The right-most column considers the finite-field case where the characteristic is the 31-bit Mersenne prime. In this specific case, the radix-8 block requires zero multiplications due to the unique structure of the 8th root-of-unity in the M31 Complex extension [SD24].

	DCCT	RDFT Radix-2	M31 RDFT Radix-8
FFT	$\frac{N}{2} \log_2 N$	$\frac{3N}{4} \log_2 N$	$\frac{7N}{16} \log_2 N$

Table 1: The number of required multiplications for FFT in different scenarios.

As shown, the concrete multiplicative complexity of RDFT is comparable to DCCT, particularly in the case of M31. However, RDFT retains all the well-known properties of Fourier transforms, notably the ability to work with high radixes—a crucial property for efficient memory management in massively parallel computing devices such as GPUs.

Acknowledgement

The author extends deep gratitude to Leo Lerer and Ilya Lesokhin from Starkware for their thorough and insightful reviews, and for pointing out the relevance of Laurent polynomials. Special thanks go to my teammates at Ingonyama's Research Team—Tomer Solberg, Karthik Inbasekar, and Suyash Bagad—for countless hours of productive discussions. Lastly, heartfelt thanks to Omer Shlomovits for his guidance unwavering support.

References

[Bha11] Gaurav Bhatnagar. In praise of an elementary identity of euler. *arXiv preprint arXiv:1102.0659*, 2011.

[Bôc95] Maxime Bôcher. Gauss's third proof of the fundamental theorem of algebra. 1895.

[Bri88] E Oran Brigham. *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[BSBHR18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[BSBHR18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, 2018.

[Dom23] Yuval Domb. *NTT 201 - Foundations of NTT hardware design*. 2023. `https://github.com/ingonyama-zk/papers/blob/main/ntt_201_book.pdf`.

[Gab24] Ariel Gabizon. From airs to raps - how plonk-style arithmetization works. `https://hackmd.io/@aztec-network/plonk-arithmetiization-air#fn1`, 2024.

[HLN23] Ulrich Haböck, Daniel Lubarov, and Jacqueline Nabaglo. Reed-solomon codes over the circle group. *Cryptology ePrint Archive*, 2023.

[HLP24] Ulrich Haböck, David Levit, and Shahar Papini. Circle starks. *Cryptology ePrint Archive*, 2024.

[İC17] Murat Burhan İlter and Murat Cenk. Efficient big integer multiplication in cryptography. *International Journal of Information Security Science*, 6(4):70–78, 2017.

[Kal84] Dan Kalman. The generalized vandermonde matrix. *Mathematics Magazine*, 57(1):15–21, 1984.

[Lan05] Serge Lang. *Undergraduate algebra*. Springer Science & Business Media, 2005.

[RS60] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[SD24] Tomer Solberg and Yuval Domb. Mersenne 31 polynomial arithmetic. `https://github.com/ingonyama-zk/papers/blob/main/Mersenne31_polynomial_arithmetic.pdf`, 2024.

[Tea23] StarkWare Team. ethstark documentation–version 1.2. Technical report, IACR preprint archive 2023, 2023.