Block-Accumulate Codes: Accelerated Linear Codes for PCGs and ZK

Vladimir Kolesnikov[1], Stanislav Peceny[2⋆], Rahul Rachuri[3], Srinivasan Raghuraman[4], Peter Rindal[5], and Harshal Shah[3]

[1] Georgia Institute of Technology
[2] Stealth Software Technologies, Inc.
[3] Visa Research
[4] Visa Research and MIT
[5] Category Labs

Abstract. Linear error-correcting codes with fast encoding and high minimum distance are a central primitive across modern cryptography. They appear prominently in at least two domains: (1) pseudorandom correlation generators (PCGs), which enable sublinear-communication generation of correlations such as oblivious transfer and vector oblivious linear evaluation, and (2) zero-knowledge proof systems, where linear-time encoders underpin proof soundness and scalability. In both settings, the prover or sender must multiply by a large generator matrix $\mathbf{G}$, often with dimensions in the millions, making computational efficiency the dominant bottleneck.

We propose a generalized paradigm for building crypto-friendly binary codes with provable minimum distance. Roughly speaking, these codes are based on randomized turbo codes such as repeat-accumulate codes. We prove linear asymptotic minimum distance and compute the exact expected weight spectrum for concrete sizes. We observe that our codes approach the Gilbert–Varshamov distance bound and outperform prior constructions.

We construct several novel codes, the most promising of which we call Block-Accumulate codes. Among codes with provable distance, our code is 8× faster than the state of the art on a CPU and 50× faster on a GPU; even against aggressive parameters with conjectured distance, it is 3× and 20× faster, respectively. Under these parameters, this yields overall PCG speedups of 2.5× on the CPU and 15× on the GPU, achieving a projected 200+ million OTs per second, or about 100 million binary Beaver triples per second, on the GPU (excluding the one-time 10 ms GGM seed expansion). We also observe a 2× encoding speedup and half the peak memory consumption in the Blaze zero-knowledge (PCS) scheme of Brehm et al. (EUROCRYPT '25).

1 Introduction

Linear error-correcting codes with fast encoding and provable minimum distance are a fundamental primitive in cryptography. Beyond their classical role in com-

⋆ Part of this work was done while the author was at Visa Research and Georgia Tech.

munication, such codes increasingly appear as computational building blocks inside larger cryptographic protocols, where encoding must be performed on vectors of size ranging from millions to billions of symbols. In these settings, the asymptotic encoding complexity and concrete constant factors of the generator matrix dominate performance, often outweighing all other cryptographic costs.

Coding-Theoretic Codes vs. Cryptographic Codes. As discussed in greater detail below, we require $\mathbf{G}$ to support fast matrix multiplication, i.e., computing $\mathbf{G} \cdot \mathbf{a}'$, and to have a high minimum distance. These requirements align with those of a generator matrix for an error-correcting code. Thus, one might simply consult the coding theory literature and select a state-of-the-art code. However, this would result in suboptimal performance. Indeed, typical coding-theoretic constructions optimize for fast encoding of moderately sized codes (on the order of thousands), whereas we are interested in code lengths on the order of millions. More importantly, coding-theoretic constructions have traditionally required an efficient decoder; that is, given $\mathbf{c}' = \mathbf{x}\mathbf{G} + \mathbf{e}$ for sparse $\mathbf{e}$, one must decode $\mathbf{c}'$ to recover $\mathbf{x}$. In contrast, codes used in cryptography typically never rely on a decoder and therefore are not subject to this constraint. Instead, we require $\mathbf{G}$ to have a concretely high minimum distance and efficient encoding. While the minimum distance upper-bounds the decodability of a code, codes with high minimum distance often do not admit efficient decoding algorithms.

Prior cryptographic code constructions, such as [CRR21,BCG+22,RRT23], have leveraged this fact by adapting constructions from coding theory to amplify minimum distance, albeit at the expense of worse decodability. One can view our work as continuing this trend, but with the added criterion that our codes should be GPU-friendly.

1.1 Applications

Correlated Randomness and MPC. Correlated randomness is the de facto foundation for realizing efficient multiparty computation (MPC). Primary examples are oblivious transfer (OT), vector-oblivious linear evaluation (VOLE), and Beaver triple correlations, which drive a large portion of today's MPC constructions. Consequently, the ability to generate such correlated randomness with low overhead is crucial for practical deployment. The development of pseudorandom correlation generators (PCGs) [BCG+19a,BCG+19b] based on linear codes with high minimum distance has been proposed as a promising direction: they offer sublinear communication complexity and compelling computational overheads. However, more research is needed to fully harness these benefits, including a deeper understanding of the underlying security assumptions and improvements in computational performance.

Interactive Zero-Knowledge Proofs and Linear Codes. The same code-based transformation central to PCGs also arises in state-of-the-art zero-knowledge (ZK) protocols. Many modern interactive proof systems, especially those built from linear-time interactive proofs and VOLE-based proof compilers [WYKW21]

[YWL+20], require the prover to encode vectors using a linear error-correcting code with high minimum distance. This encoding step serves the dual purpose of ensuring soundness (via distance properties) and enabling linear-time arithmetic checks. In these settings, the prover's bottleneck is the cost of multiplying by a large generator matrix $\mathbf{G}$, often of size in the millions. Thus, the ZK literature has emphasized the need for fast, linear-time encoders for codes with strong distance guarantees. Our constructions directly satisfy this need: they drop into existing interactive ZK frameworks, replacing generic coding-theory choices with GPU-friendly codes that deliver linear-time encoding.

Codes in SNARKs and IOPs. Beyond interactive proofs, linear codes with efficient encoding are also a cornerstone of SNARK and STARK systems. Examples include the FRI protocol and its descendants in STARKs [BSBHR19], Ligero [AHIV17], Aurora [BSCR+19], and more recent transparent SNARKs such as Brakedown [GLS+23], Plonkish systems [GWC19], and Spartan [Set20]. In all of these schemes, prover efficiency is tied to encoding large vectors under a code with strong distance guarantees, often with dimensions in the millions. Recently, several works have pursued designing or applying similar linear codes for this setting [BFK+24,LZ25,BMMS26], notably Blaze [BCF+25], whose efficiency we improve upon in this work. As in the interactive setting, the dominant cost is multiplying by a generator matrix $\mathbf{G}$, and thus these systems explicitly benefit from linear-time encoders. Our codes can drop directly into such SNARK frameworks, offering provable distance and substantial speedups. We note that some ZK works [BFK+24] focus on the non-binary case while we focus on binary codes. Our enumeration techniques can be generalized to this setting by scaling the relevant probabilities by one over the field size. We leave a detailed non-binary analysis to future work.

See Appendix A for a more detailed handling of these applications.

1.2 Contribution

We make the following contributions for binary linear codes over $\mathbb{F}_2$:

- New state-of-the-art linear codes for PCGs and zero knowledge that are several times faster than prior art at rate 1/2. On a CPU, our linear-code encoding step requires about 20 ms for $k = 2^{20}$ outputs and is between 3× to 8× faster than prior art, depending on the parameterization.
- GPU-accelerated implementation that takes advantage of the parallel structure of our codes. On a GPU, our code requires just 3.2 ms for $k = 2^{20}$ outputs and is approximately 50× faster than prior art. This corresponds to a throughput of roughly 200 million OTs per second, or roughly 100 million binary Beaver triples per second.
- We integrate our new codes into the binary zero knowledge system Blaze [BCF+25]. As a proof of concept, we replace their rate 0.25 repeat-accumulate-accumulate (RAA) codes with our new construction and observe a 2× speedup in encoding time and a halving of total memory usage.

– A flexible and extensible framework for proving the minimum distance of codes. In addition to our codes, we anticipate that this framework can be used to prove the minimum distance of a wide variety of codes, possibly resulting in further improvements.
– Improved minimum distance compared to prior work. This means that when deployed in a PCG protocol, smaller-weight noise vectors can be used to achieve the same security, resulting in less communication.

2 Related Work

We now review techniques aimed at optimizing PCGs. Excluding recent works based on Ring LPN [BCG+20,BBC+24,LXYY25], most PCGs rely on essentially the same protocols [BCGI18,BCG+19a,BCG+19b,BCG+20,CRR21,BCG+22] [RRT23,KPRR25]: they generate a secret sharing of the product $[\![\Delta \mathbf{a}']\!]$, where $\mathbf{a}' \in \mathbb{F}^n$ is a sparse vector and $\Delta \in \mathbb{F}$ is a scalar. This corresponds to the interactive setup $\mathbf{c}' - \mathbf{b}' = \Delta \mathbf{a}'$ (see Section 1). The resulting shared vector is then compressed using a linear function $\mathbf{G}$. Consequently, improving PCG performance centers on two tasks: (1) generating the shared sparse vector $[\![\Delta \mathbf{a}']\!]$ efficiently, and speeding up the subsequent multiplication by $\mathbf{G}$.

Two major optimizations have emerged for generating the secret sharing of $\Delta \mathbf{a}'$. The first is the concept of a regular distribution for $\mathbf{a}'$ [AFS05], where $\mathbf{a}'$ is the concatenation of t unit vectors of length n/t. That is, n is the length of $\mathbf{a}'$ and t is its Hamming weight. Regular noise replaced the more traditional Bernoulli and exact distributions [AFS05,HOSS18,Ds17,BCGI18,SGRR19,BCG+19a] [BCG+19b,BCG+20,BCG+22]. In the context of secure computation, this approach was first used by TinyKeys [HOSS18] and subsequently by [BCGI18] for PCGs. The regular noise distribution is advantageous because it allows generating $[\![\Delta \mathbf{a}']\!]$ with only $O(n)$ AES calls, spread across t distributed point functions. This is in contrast with the other noise distributions, which typically require $O(t)$ distributed point functions, each of size n, resulting in $O(tn)$ AES calls. Alternatively, one can use more complex techniques such as cuckoo hashing [SGRR19], but these also impose additional overheads.

More recently, [KPRR25] presented a new assumption that allows the cost of generating $[\![\Delta \mathbf{a}']\!]$ to be amortized across q instances by fixing the noisy locations of $\mathbf{a}'$. This allows for many correlated noise vectors $\mathbf{a}'^{(1)}, ..., \mathbf{a}'^{(q)} \in \mathbb{F}^n$ to be generated more efficiently. As long as the sparse noise elements in each $\mathbf{a}'^{(i)}$ are sampled uniformly at random, the resulting $\mathbf{a}^{(i)} := \mathbf{G}\mathbf{a}'^{(i)} \in \mathbb{F}^k$ will be pseudorandom by the stationary LPN assumption [KPRR25]. This leads to a concrete 6–18× reduction in communication along with compelling performance improvements for our GPU implementation.

Another line of research, which constitutes the principal runtime bottleneck for PCGs and is the focus of this work, aims to reduce the cost of multiplying $\mathbf{G}$ by $[\![\Delta \mathbf{a}']\!]$ [BCGI18,BCG+19a,BCG+19b,BCG+20,CRR21,BCG+22,RRT23]. In the original formulation of Syndrome Decoding and LPN, $\mathbf{G}$ is taken to be uniform, so computing $\mathbf{G} \cdot [\![\Delta \mathbf{a}']\!]$ requires $O(n^2)$ time. However, there is good

evidence that one may replace $\mathbf{G}$ with the generator matrix of a code that has high minimum distance, unless that code exhibits a strong algebraic structure (e.g., a Reed–Solomon code with efficient Syndrome Decoding via Berlekamp–Massey [Ber68]). Switching $\mathbf{G}$ to a generator matrix of a linear-time code makes it theoretically possible to evaluate $\mathbf{G} \cdot \mathbf{v}$ in $O(n)$ time for any $\mathbf{v}$. In practice, however, performance often degrades to $O(cn)$, where c is typically related to $\log n$ or the security parameter. Several works [BCGI18,BCG+19a,BCG+19b] propose using LDPC or quasi-cyclic codes, which offer well-studied security and reasonable performance. Subsequently, [BCG+22] introduced a turbo-like code, called Expand-Accumulate, which significantly reduces costs and can be parameterized to achieve a provably high minimum distance. Further improvements were proposed by [RRT23], who replaced [BCG+22]'s accumulation step with a convolution, thereby boosting minimum distance and enabling more favorable parameters. However, none of these existing designs focus on codes suitable for efficient GPU acceleration. Moreover, existing proof techniques do not result in codes with optimal minimum distance. When translated into the context of PCGs, low minimum distance means that the setup phase must be more expensive to maintain security. Our primary objective, therefore, is to develop a code with extremely high minimum distance that can be implemented efficiently on both CPUs and GPUs.

In the ZK literature, some additional works such as [BFK+24,BCF+25] generalize RA codes [BMS09] to work over larger fields. We note that our codes could be generalized to the non-binary case using our core techniques.

3 Technical Overview

We now provide a high-level overview of our key techniques and improvements. In Section 5, we formally introduce our enumeration framework, which enables flexible analysis of the minimum distance of various turbo-like codes and their concatenations. This framework makes it easy to evaluate new code constructions and identify suitable candidates for PCGs. We believe the codes we propose are only a starting point and that further improvements are likely. In Section 6 - 9, we observe that existing state-of-the-art codes for PCGs are neither constant depth nor amenable to parallel implementations, which limits their ability to exploit modern GPU architectures. Guided by our enumeration framework, we introduce and analyze a novel Block-Diagonal code (Section 6) that is constant depth, highly parallelizable, and therefore well suited for efficient GPU execution. In Section 7, we then combine Block-Diagonal codes with Accumulators, again using our framework, to obtain our final construction, which outperforms previous schemes on both CPUs and GPUs. In Section 9, we discuss a potential alternative to the Block-Diagonal code, called the Banded-Diagonal code, which offers several potential advantages. Finally, in Section 10, we experimentally evaluate the performance of our newly proposed codes.

More specifically:

– **Turbo-like Codes.** Our work considers turbo-like codes [BGT93,BMS09]. Turbo codes can take many forms in general, but we focus on the serial concatenation of subcodes $\mathbf{G}_1, \ldots, \mathbf{G}_t$, where the input $\mathbf{x}$ is first multiplied by the matrix $\mathbf{G}_1$, then permuted by a random permutation π_1, and subsequently multiplied by the next subcode $\mathbf{G}_2$, and so on. I.e., our overall code can be expressed as

$$\mathbf{G} := \mathbf{G}_1 \cdot \pi_1 \cdot \mathbf{G}_2 \cdot \ldots \cdot \pi_{t-1} \cdot \mathbf{G}_t.$$

This genre of code is called a serially-concatenated turbo code of depth t. Conditioned on each $\mathbf{G}_i$ satisfying certain properties, we next present a generic method for proving the expected minimum distance. Our approach can be viewed as a generalization of [BMS09].

– **Enumeration Framework.** To prove that our codes have linear minimum distance, we present a flexible enumeration framework. We generalize the idea of an *enumerator* [KU98,BMS09], a technique that has been used to prove the distance of various codes. In particular, we introduce new combinatorial techniques to compute the expected minimum distance for families of structured random codes.

Essentially, for each step of our code, i.e. for each multiplication by $\mathbf{G}_i$, we track the current expected distribution of $\{\mathbf{x}\mathbf{G}_1\pi_1\mathbf{G}_2 \ldots \pi_{i-1}\mathbf{G}_i \mid \mathbf{x} \in \mathbb{F}^k\}$ with respect to some property. Typically, this will be the expected number of codewords with each Hamming weight. However, at times we consider other properties as well. We believe this paradigm is likely to be useful for proving tighter bounds on minimum distance for other codes as well, such as the prior Expand-Accumulate [BCG+22] and Expand-Convolute codes [RRT23].

– **Enumerator Implementation.** The prior work of [BMS09] derives exact weight enumerators and then uses analytic bounds on them to establish asymptotic minimum-distance guarantees. In contrast, we directly evaluate the exact finite-length enumerators for each component of our codes to obtain the expected minimum distance. More generally, we compute the whole weight distribution of the codewords in a fine-grained manner, not just the expected minimum distance. This in turn allows us to more accurately compute the security level of LPN in the linear test framework. However, this fine-grained process can be computationally intensive, which required implementing low-level optimizations and a new formulation for how to compose the enumerators from each step.

– **Block-Diagonal Codes.** We give concrete instantiations for the $\mathbf{G}_i$ matrices. The first is referred to as the Block-Diagonal code and is presented in Section 6. Each $\mathbf{G}_i$ is sampled with q uniformly random submatrices along the diagonal, each having $\sigma := O(n/q)$ rows and columns. I.e.,

$$\mathbf{G}_i = \begin{pmatrix} \mathbf{G}_{i,1} & \mathbf{O} & \ldots & \mathbf{O} \\ \mathbf{O} & \mathbf{G}_{i,2} & \ldots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \ldots & \mathbf{G}_{i,q} \end{pmatrix},$$

where $\mathbf{G}_{i,j} \in \mathbb{F}^{O(\sigma)\times O(\sigma)}$. We show that this construction achieves linear minimum distance with good constants with sublinear σ and small t. Moreover, our construction differs from most turbo codes in that it is based on a "stateless" non-recursive convolution. For our purposes, this essentially means that the Block-Diagonal construction is easily implemented on a GPU in a highly parallel manner.
For constant t we show that this code achieves linear minimum distance using bounding techniques. Via our enumeration techniques we observe that t iterations and a memory size of $\sigma = \sqrt[t]{n}$ is sufficient for distance approaching the GV bound. For the asymptotic case of $n \to \infty$ and constant t we show linear minimum distance. We conjecture $\sigma = \sqrt[t]{n}$ is a general trend for non-constant t given that the distance of $\mathbf{G}_i$ does not collapse to 0. However, this case remains open.

- **Banded-Diagonal Codes.** We give a second code referred to as the Banded-Diagonal construction, where $\mathbf{G}_i$ consists of a band of random values along the diagonal, i.e. no square submatrices. We believe that this construction could offer certain advantages compared to the Block-Diagonal construction and also achieves linear minimum distance. We present this construction at a high level in Section 9 (and its enumerator in Appendix G).
- **Block-Accumulate Codes.** Our most promising code replaces the repetition layer of repeat-accumulate codes with a random Block-Diagonal subcode. We refer to this code as Block-Accumulate; it has excellent CPU/GPU performance and concrete distance approaching that of a random linear code. This code can be described as

$$\mathbf{G} = \mathbf{G}_1\pi_1\mathbf{A}...\pi_{t-1}\mathbf{A},$$

  where $\mathbf{G}_1$ is a Block-Diagonal subcode and $\mathbf{A}$ is an accumulator that performs a prefix sum of the input.
  Prior work [BMS09] showed that accumulator-based turbo codes can achieve linear minimum distance, but efficient concrete instantiations used in cryptography have typically paid for this with lower rate or less GPU-friendly structure. Our Block-Accumulate codes retain linear-time encoding, operate natively at rate 1/2, and for concrete parameters achieve distance approaching the GV bound. Asymptotically, we show linear minimum distance assuming $\mathbf{G}_1$ is instantiated with integer per-block minimum distance ≥ 5.
- **Code Implementation.** We implement several variants of our codes in CUDA and our Block-Accumulate Codes on the CPU. We observe significant speedups over prior art in both settings. We perform benchmarking on a laptop equipped with a cost-effective NVIDIA GeForce RTX 4060 GPU. For $k = 2^{20}$, our method compresses data in $\approx$ 3ms on a GPU and $\approx$ 21ms on a CPU, achieving a $\approx 47\times$ speedup over the state-of-the-art Expand-Accumulate code [BCG+22] on the GPU and $\approx 7.8\times$ speedup over the state-of-the-art Expand-Convolute code [RRT23] on the CPU.

A note on generalization. Over $\mathbb{F}_p$, our enumerator generalizes essentially "for free" for the same reason it works in the binary case: once each stage is suffi-

ciently randomized, weight propagation depends only on whether symbols are zero or nonzero, not on their specific values. Concretely, we replace binary mixing with $\mathbb{F}_p$-linear mixing using uniformly random coefficients (and, where structured components appear, most notably accumulators, we insert inexpensive random *weighters*, i.e., random diagonal scalings in $\mathbb{F}_p^*$, before/after the accumulator, see Rosnes et al. [RGiA11]). This standard symmetrization prevents value-dependent cancellations and ensures that, conditioned on the input support, nonzero symbols are close to uniformly distributed in $\mathbb{F}_p^*$. With this in place, the analysis carries over by the same composition rules as in the binary setting, with the only quantitative change being the replacement of "1/2" effects by the corresponding $\mathbb{F}_p$ constants (e.g., uniformity gives $\Pr[\text{symbol} = 0] = 1/p$ and $\Pr[\text{symbol} \neq 0] = 1 - 1/p$). However, we do not analyze this setting in detail nor prove its linear distance. We leave it to future work.

4 Preliminaries

4.1 Notation

Let $\mathbb{Z}_{\geq 0} := \{0, 1, 2, ...\}$ and $\mathbb{N} := \{1, 2, 3, ...\}$. For any $n \in \mathbb{Z}_{\geq 0}$, $[n] := \{1, 2, \ldots, n\}$ denotes the set of natural numbers $1, \ldots, n$ (both inclusive). $[0]$ is defined to be the empty set $\emptyset$. For any $a \leq b \in \mathbb{Z}_{\geq 0}$, $[a, b]$ denotes the set of integers $a, \ldots, b$ (both inclusive). For any $a > b \in \mathbb{Z}_{\geq 0}$, $[a, b]$ is defined to be the empty set $\emptyset$.

For any vector $\mathbf{x} \in \{0,1\}^k$, where $k \in \mathbb{Z}_{\geq 0}$, let $x_i \in \{0, 1\}$ denote the ith element of $\mathbf{x}$ for $i \in [k]$, and let $|\mathbf{x}| := k$ denote its length. For any $S \subseteq \mathbb{N}$, let $\mathbf{x}_S = \{x_i : i \in S \cap [k]\}$ denote the subvector of $\mathbf{x}$ indexed by the indices that belong to S. For any vector $\mathbf{x}$, we define $|\mathbf{x}|_{\mathsf{H}} := |\{i : x_i \neq 0\}|$ as the number of nonzero elements of $\mathbf{x}$.

4.2 Combinatorial Functions.

Balls in bins. There are

$$\mathsf{B}(n, k) = \binom{n+k-1}{k-1} = \left| \left\{ \mathbf{x} \in \mathbb{N}^k \mid \sum_i \mathbf{x}_i = n \right\} \right|$$

ways to assign $n \in \mathbb{N}$ identical balls into $k \in \mathbb{N}$ labeled bins. This can be easily seen from the following argument. Consider a binary vector $\mathbf{b} \in \{0,1\}^{n+k-1}$ where each position represents a "slot." Suppose we designate $k-1$ of them as "demarcators" with value 1, partitioning the remaining n slots into k groups, where each group is a sequence of zeros. The size of each group represents the number of balls (out of n) allocated to each of the k bins. Each choice of demarcators results in a unique way of assigning the n identical balls into k labeled bins. Since there are $\binom{n+k-1}{k-1}$ ways to choose the demarcators, i.e. a binary string $\mathbf{b}$ with weight $k-1$, the claim follows. The very same argument also shows that the number of $\mathbf{x}$ s.t. $\sum_{i \in [k]} \mathbf{x}_i = n$ is $\mathsf{B}(n, k) = \binom{n+k-1}{k-1}$.

Balls in bins w/ capacity. There are

$$\begin{aligned}\mathsf{C}(n,k,c) &= \sum_{i=0}^{k}(-1)^i\binom{k}{i}\binom{n+k-i(c+1)-1}{k-1} \\ &= \left|\left\{\mathbf{x}\in\mathbb{N}^k \mid \sum_i \mathbf{x}_i = n \wedge \mathbf{x}_i \leq c\right\}\right|\end{aligned}$$

ways to assign n identical balls into k labeled bins each having a maximum capacity of c. A proof of this result can be found in [Bro19].

Balls in slotted bins. There are

$$\begin{aligned}\mathsf{S}(n,k,c) &= \sum_{i=0}^{k}(-1)^i\binom{k}{i}\binom{c(k-i)}{n} \\ &= \left|\left\{\mathbf{x}\in\{0,1\}^{k\times c} \mid \forall i\in[k], 0 < \sum_j \mathbf{x}_{i,j} \leq c \wedge \sum_{i,j}\mathbf{x}_{i,j} = n\right\}\right|\end{aligned}$$

ways to assign n identical balls into $k \times c$ labeled slots such that each set of c consecutive slots (i.e. each $\mathbf{x}_i$, called a "slotted bin" of size c) has at least one ball. This can be proven using an inclusion-exclusion style counting argument. See Appendix C.

4.3 Coding Theory

Generator Matrix. Let $k < n$, where $n, k \in \mathbb{N}$, and let $\mathbf{G} \in \mathbb{F}^{k\times n}$ be a matrix over the field $\mathbb{F}$. The matrix $\mathbf{G}$ is called the generator matrix of the linear code $\mathcal{C}$, which is defined as $C \coloneqq \{\mathbf{c} = \mathbf{x}\mathbf{G} \mid \mathbf{x} \in \mathbb{F}^k\}$. Here, $\mathbf{x}$ represents an arbitrary message vector of length k, and $\mathbf{c}$ is the corresponding codeword in $\mathcal{C}$.

Systematic Form. A common representation for the generator matrix is its *systematic form*, expressed as $\mathbf{G} = \left[\mathbf{I}_k||\mathbf{P}\right]$, where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{P}$ is a $k \times (n-k)$ matrix. When $\mathbf{G}$ is in systematic form, the code $\mathcal{C}$ is said to be *systematic* since the original message appears directly in the first k positions of the codeword $\mathbf{c} = \mathbf{x}\mathbf{G} = \left[\mathbf{x}||\mathbf{p}\right]$, with $\mathbf{p} \in \mathbb{F}^{n-k}$ representing the parity information. Any generator matrix $\mathbf{G}'$ can be transformed into systematic form by performing elementary row operations and, if necessary, column permutations. This preserves the structural properties of the code, such as the minimum distance.

Parity Check Matrix. A matrix $\mathbf{H} \in \mathbb{F}^{(n-k)\times n}$ is called a parity check matrix of $\mathcal{C}$ if it represents a linear function $\varphi : \mathbb{F}^n \to \mathbb{F}^{n-k}$ whose kernel is exactly $\mathcal{C}$. Equivalently, we have $C = \{\mathbf{c} \in \mathbb{F}^n \mid \mathbf{H}\mathbf{c}^T = \mathbf{0}\}$. It follows that $\mathbf{H}$ has dimensions $(n-k) \times n$. The code defined by $\mathbf{H}$ is known as the *dual code* of $\mathcal{C}$.

Minimum Distance. The minimum distance d of a linear code $\mathcal{C}$ is defined as the smallest number of positions in which any two distinct codewords differ. Equivalently, since $\mathcal{C}$ is a linear code, d is equal to the minimum weight (i.e., the number of nonzero entries) of any nonzero codeword in $\mathcal{C}$. Another equivalent definition is that d is the minimum number of linearly dependent columns of the parity check matrix $\mathbf{H}$. It is known that computing the minimum distance for an arbitrary linear code is an NP-complete problem.

Relationship Between Generator and Parity Check Matrices. There is a direct connection between generator matrix and parity check matrix. In particular, if the generator matrix is in systematic form, $\mathbf{G} = \left[\mathbf{I}_k || \mathbf{P}\right]$, then a corresponding parity check matrix for $\mathcal{C}$ can be constructed as $\mathbf{H} = \left[-\mathbf{P}^T || \mathbf{I}_{n-k}\right]$, where $\mathbf{P}^T$ denotes the transpose of $\mathbf{P}$, and $\mathbf{I}_{n-k}$ is the $(n-k) \times (n-k)$ identity matrix.

5 The Enumeration Framework

Parallel Turbo Codes. Turbo codes are an extremely successful class of linear error-correcting codes [BGT93,VY12], known for their low encoding complexity and ability to correct errors near the channel capacity. Classical turbo code constructions typically use a parallel concatenation of recursive convolutional constituent encoders, each applied to a differently interleaved (i.e., permuted) copy of the input. More specifically, a message $\mathbf{x} \in \mathbb{F}^k$ is encoded to a codeword $\mathbf{c} = (\mathbf{c}_1 || ... || \mathbf{c}_t)$, where the subcodeword $\mathbf{c}_i \in \mathbb{F}^\sigma$ for $\sigma := n/t$ is computed as

$$\mathbf{c}_i := \mathbf{x}\pi_i\mathbf{G}_i,$$

where π_i is a random permutation matrix, and $\mathbf{G}_i \in \mathbb{F}^{k \times \sigma}$ is a recursive convolution. $\mathbf{G}_i$ is typically a structured upper triangular matrix known as a convolution. In particular, for each $\mathbf{x} \in \mathbb{F}^k$, $\mathbf{c} := \mathbf{x}\mathbf{G}_i$ can be computed iteratively: at the j-th iteration, $\mathbf{c}_j$ is computed as a linear combination of $\mathbf{x}_j$ and some internal state vector $\mathbf{s}_j \in \mathbb{F}^m$. Typically, one defines $\mathbf{s}_j := (\mathbf{c}_{j-1}, \ldots, \mathbf{c}_{j-m})$ as the vector of the previous m outputs. m is referred to as the state size or memory size of the convolution. These codes are called recursive convolutional codes because each output bit is computed as a linear combination of the current input bit and the previous m output bits.

While turbo codes with parallel recursive convolutions achieve good encode-decode performance in practice, it has been proven that they cannot achieve constant rate and linear minimum distance unless m is linear in n [KU98,BMS09]. This parallel structure is often preferred because it facilitates efficient decoding.

Serial Turbo Codes. An alternative to parallel codes are serially concatenated codes [KU98,BDMP98], where the output of the previous subcode is the input to the next. I.e., $\mathbf{c}_0 := \mathbf{x}$, $\mathbf{c}_1 := \mathbf{c}_0\mathbf{G}_1$, and $\mathbf{c}_i := \mathbf{c}_{i-1}\pi_{i-1}\mathbf{G}_i$ for $i \in [2, t]$. For $t = 3$, [KU98,BMS09] proved that the repeat-accumulate-accumulate construction (see Section 5.1, accumulator is a particularly simple convolution), the code achieves a concretely high linear minimum distance at rate $1/4$. Unfortunately, for efficiency

reasons we are interested in rate $1/2$, where the concrete minimum distance is low. In the coding theory literature, all $\mathbf{c}_i$ are typically included in the final output to facilitate decoding, where $\mathbf{c}_i$ are the parity bits of $\mathbf{c}_{i-1}$. However, it is the last subcodeword $\mathbf{c}_t$ that contributes the most to minimum distance. Indeed, this is implicit in [BMS09], which considers only $\mathbf{c}_t$ in the proof and writes $\mathbf{c} := \mathbf{x}\mathbf{G}_1\pi_1\mathbf{G}_2 \cdots \pi_{t-1}\mathbf{G}_t$.

5.1 Repeat-Accumulate Codes

The most prominent serial turbo code is the repeat-accumulate (RA) code. Introduced in [DJM98,KU98], the RA code was later generalized from $t = 2$ to greater depths t, yielding variants such as the repeat-accumulate-accumulate (RAA) code or, more generally, the repeat-accumulate-t (RA-t) code. The RA code construction is conceptually simple and follows the outline above. Let r denote the code rate. The matrix $\mathbf{G}_1$ is defined to repeat the input $\mathbf{x}$ exactly $e := 1/r$ times; that is, $\mathbf{G}_1 := \mathbf{R} := (\mathbf{I}_k \,||\, \cdots \,||\, \mathbf{I}_k)$, where $\mathbf{I}_k \in \{0,1\}^{k\times k}$ is the identity matrix. For $i \in [2,t]$, the convolution matrix $\mathbf{G}_i := \mathbf{A}$, where $\mathbf{A}$ is an accumulator, an upper triangular matrix with all ones on and above the diagonal. Multiplication by $\mathbf{A}$ is a straightforward prefix sum on the input vector, i.e. $(\mathbf{x}\mathbf{A})_i = \mathbf{x}_1 + \mathbf{x}_2 + ... + \mathbf{x}_i$. The encoding of a message $\mathbf{x}$ into a codeword $\mathbf{c}$ is defined by

$$\mathbf{c} = \mathbf{x}\mathbf{R}\pi_1\mathbf{A}...\pi_{t-1}\mathbf{A}$$

As shown in [BMS09], when $t = 3$, $e = 4$, and the permutations π_1, π_2 are chosen uniformly at random, the resulting RAA code achieves a concretely high linear minimum distance with good probability. We will discuss this approach later.

5.2 Expected Enumerators

Let $\mathbf{G} \in \{0,1\}^{k\times n}$ be the generator matrix of a code mapping k-bit input words into n-bit output codewords, where $k \le n$. Let $\mathcal{D}$ denote a distribution over $\{0,1\}^{k\times n}$ and suppose we sample $\mathbf{G}$ according to $\mathcal{D}$. Then we define the input-output Hamming weight enumerator as:

$$\mathsf{E}^{\mathcal{D}}_{w,h} := \mathbb{E}_{\mathbf{G}\leftarrow\mathcal{D}} \left|\left\{\mathbf{x} \in \mathbb{F}^k \mid |\mathbf{x}|_{\mathsf{H}} = w, |\mathbf{x}\mathbf{G}|_{\mathsf{H}} = h\right\}\right|.$$

That is, $\mathsf{E}^{\mathcal{D}}_{w,h}$ is the expected number of input words $\mathbf{x} \in \{0,1\}^k$ with $|\mathbf{x}|_{\mathsf{H}} = w$ and the corresponding codeword $\mathbf{c} = \mathbf{x}\mathbf{G}$ with $|\mathbf{c}|_{\mathsf{H}} = h$. It will be useful to let $E^{\mathcal{D}}$ be the random variable that $\mathsf{E}^{\mathcal{D}}$ is the expectation of, i.e.

$$E^{\mathcal{D}}_{w,h} := \left|\left\{\mathbf{x} \in \mathbb{F}^k \mid |\mathbf{x}|_{\mathsf{H}} = w, |\mathbf{x}\mathbf{G}|_{\mathsf{H}} = h\right\}\right| \quad \text{where} \quad \mathbf{G} \leftarrow \mathcal{D}.$$

Hence, $\mathsf{E}^{\mathcal{D}}_{w,h} = \mathbb{E}_{\mathbf{G}\leftarrow\mathcal{D}}[E^{\mathcal{D}}_{w,h}]$. Let $X_{\mathbf{x},\mathbf{G},h}$ be the indicator random variable that equals 1 if the codeword $\mathbf{c} = \mathbf{x}\mathbf{G}$, corresponding to the input $\mathbf{x} \in \{0,1\}^k$, has Hamming weight $|\mathbf{c}|_{\mathsf{H}} = h$, and 0 otherwise. We have

$$\mathsf{E}^{\mathcal{D}}_{w,h} = \sum_{\substack{\mathbf{x}\in\{0,1\}^k,\\ |\mathbf{x}|_{\mathsf{H}}=w}} \Pr_{\mathbf{G}\leftarrow\mathcal{D}}\left[X_{\mathbf{x},\mathbf{G},h} = 1\right].$$

Similarly, let $X_{\mathbf{x},\mathbf{G},\mathbf{y}}$ be the indicator random variable that equals 1 if $\mathbf{y} = \mathbf{x}\mathbf{G}$, and 0 otherwise. We have

$$X_{\mathbf{x},\mathbf{G},h} = \sum_{\substack{\mathbf{y}\in\{0,1\}^n \\ |\mathbf{y}|_{\mathsf{H}}=h}} X_{\mathbf{x},\mathbf{G},\mathbf{y}},$$

and therefore

$$\mathsf{E}^{\mathcal{D}}_{w,h} = \sum_{\substack{\mathbf{x}\in\{0,1\}^k,\mathbf{y}\in\{0,1\}^n, \\ |\mathbf{x}|_{\mathsf{H}}=w,|\mathbf{y}|_{\mathsf{H}}=h}} \Pr_{\mathbf{G}\leftarrow\mathcal{D}}\left[X_{\mathbf{x},\mathbf{G},\mathbf{y}} = 1\right].$$

5.3 Serially Concatenated Enumerators

Let $\mathbf{G}_1 \in \{0,1\}^{k\times n_1}, \mathbf{G}_2 \in \{0,1\}^{n_1\times n}$ be generator matrices of a code that maps a k-bit word to an n_1-bit intermediate encoding, and then to an n-bit codeword. Let $\mathcal{D}_1, \mathcal{D}_2$ denote the distributions from which $\mathbf{G}_1, \mathbf{G}_2$ are sampled, respectively. Let $\pi \in \{0,1\}^{n_1\times n_1}$ be a permutation matrix acting on n_1-bit words, and let $\mathcal{D}_\pi$ denote the uniform distribution over all such permutation matrices. We now consider the generator matrix $\mathbf{G} \in \{0,1\}^{k\times n}$ sampled as

$$\mathbf{G} := \mathbf{G}_1\pi\mathbf{G}_2, \quad \text{where} \quad \mathbf{G}_1 \leftarrow \mathcal{D}_1, \mathbf{G}_2 \leftarrow \mathcal{D}_2, \pi \leftarrow \mathcal{D}_\pi,$$

to be the generator matrix of a code mapping k-bit input words into n-bit output codewords. Let $\mathcal{D} := [\mathbf{G}_1\pi\mathbf{G}_2 \mid \mathbf{G}_1 \leftarrow \mathcal{D}_1, \pi \leftarrow \mathcal{D}_\pi, \mathbf{G}_2 \leftarrow \mathcal{D}_2]$.

We seek to express $\mathsf{E}^{\mathcal{D}}_{w,h}$ as a composition of $\mathsf{E}^{\mathcal{D}_1}_{w,h_1}$ and $\mathsf{E}^{\mathcal{D}_2}_{h_1,h}$ for $h_1 \in [n_1]$. Consider the set of input words $\mathbf{x} \in \{0,1\}^k$ with $|\mathbf{x}|_{\mathsf{H}} = w$. Then the expected number of codewords $\mathbf{c}_1 = \mathbf{x}\mathbf{G}_1$, for $\mathbf{G}_1 \leftarrow \mathcal{D}_1$, with $|\mathbf{c}_1|_{\mathsf{H}} = h_1$, is $\mathsf{E}^{\mathbf{G}_1}_{w,h_1}$. Note that over the choice of $\pi \leftarrow \mathcal{D}_\pi$, $\tilde{\mathbf{c}}_1 = \mathbf{c}_1\pi$ is a random word with $|\tilde{\mathbf{c}}_1|_{\mathsf{H}} = |\mathbf{c}_1|_{\mathsf{H}}$. Thus, consider a uniformly random input word $\mathbf{c}' \in \{0,1\}^{n_1}$ with $|\mathbf{c}'|_{\mathsf{H}} = h_1$. The probability (over the choice of $\mathbf{c}'$) that the corresponding codeword $\mathbf{c} = \mathbf{c}'\mathbf{G}_2$ has $|\mathbf{c}|_{\mathsf{H}} = h$ is

$$\Pr_{\mathbf{G}_2,\pi}[|\mathbf{c}'\mathbf{G}_2|_{\mathsf{H}} = h \mid |\mathbf{c}'|_{\mathsf{H}} = h_1] = p = \frac{E^{\mathcal{D}_2}_{h_1,h}}{\binom{n_1}{h_1}}.$$

Therefore,

$$\begin{aligned}
\mathbb{E}_{\pi\leftarrow\mathcal{D}_\pi}\left[E^{\mathcal{D}_1\pi\mathcal{D}_2}_{w,h}\right] &= \sum_{v\in\mathbb{Z}_{\geq 0}} \Pr_{\pi\leftarrow\mathcal{D}_\pi}\left[E^{\mathcal{D}_1\pi\mathcal{D}_2}_{w,h} = v\right] * v \\
&= \sum_{v\in\mathbb{Z}_{\geq 0},\mathbf{x},\mathbf{c}',\mathbf{c}} \Pr_{\mathbf{G}_1\leftarrow\mathcal{D}_1}\left[\mathbf{x}\mathbf{G}_1 = \mathbf{c}'\right] \Pr_{\mathbf{G}_2}[\mathbf{c}'\pi\mathbf{G}_2 = \mathbf{c} \mid |\mathbf{c}|_{\mathsf{H}} = v] * v \\
&= \sum_{h_1=0}^{n_1} \frac{E^{\mathcal{D}_1}_{w,h_1}}{\binom{n_1}{h_1}} \sum_{v,\mathbf{c}' \text{ s.t. } |\mathbf{c}'|_{\mathsf{H}}=h_1,\mathbf{c}} \Pr_{\mathbf{G}_2}[\mathbf{c}'\pi\mathbf{G}_2 = \mathbf{c} \mid |\mathbf{c}|_{\mathsf{H}} = v] * v \\
&= \sum_{h_1=0}^{n_1} \frac{E^{\mathbf{G}_1}_{w,h_1} \cdot E^{\mathbf{G}_2}_{h_1,h}}{\binom{n_1}{h_1}}.
\end{aligned}$$

Finally, since $\mathbf{G}_1$ and $\mathbf{G}_2$ are sampled independently,

$$\mathbb{E}_{\mathbf{G}\leftarrow\mathcal{D}}\left[E^{\mathbf{G}}_{w,h}\right] = \sum_{h_1=0}^{n_1} \frac{\mathbb{E}_{\mathbf{G}_1\leftarrow\mathcal{D}_1}\left[E^{\mathbf{G}_1}_{w,h_1}\right] \cdot \mathbb{E}_{\mathbf{G}_2\leftarrow\mathcal{D}_2}\left[E^{\mathbf{G}_2}_{h_1,h}\right]}{\binom{n_1}{h_1}},$$

and therefore we have $\mathsf{E}^{\mathcal{D}}_{w,h} = \sum_{h_1=0}^{n_1} \frac{\mathsf{E}^{\mathcal{D}_1}_{w,h_1}\mathsf{E}^{\mathcal{D}_2}_{h_1,h}}{\binom{n_1}{h_1}}$.

We first formalize the full depth-t serial ensemble as follows.

Definition 1 (Depth-t Serially Concatenated Ensemble). *Fix $t \geq 1$ and dimensions $n_0 := k, n_1, \ldots, n_t := n$. For each $i \in [t]$, let $\mathcal{D}_i$ be a distribution over matrices $\{0,1\}^{n_{i-1}\times n_i}$ and sample subcodes independently as $G_i \leftarrow \mathcal{D}_i$. For each $i \in [t-1]$, let $\pi_i \in \{0,1\}^{n_i \times n_i}$ be a uniform random permutation matrix sampled independently. The depth-t generator matrix $G \in \{0,1\}^{k\times n}$ is:*

$$G := G_1\pi_1 G_2 \cdots \pi_{t-1} G_t.$$

We write $\mathcal{D}_{1:t}$ for the induced distribution of G.

Iterative Enumeration. Consider the case of computing an enumerator over larger depth t. E.g., for $t = 3$, we have $\mathbf{G} = \mathbf{G}_1\pi_1\mathbf{G}_2\pi_2\mathbf{G}_3$. The template above would then suggest that

$$\mathsf{E}^{\mathcal{D}}_{w,h} = \sum_{h_1=0}^{n_1}\sum_{h_2=0}^{n_2} \frac{\mathsf{E}^{\mathcal{D}_1}_{w,h_1}\mathsf{E}^{\mathcal{D}_2}_{h_1,h_2}\mathsf{E}^{\mathcal{D}_2}_{h_2,h}}{\binom{n_1}{h_1}\binom{n_2}{h_2}}.$$

Calculating this sum as is requires $O(n^{t-1})$ iterations. For any t that is not a tiny constant, the runtime of computing the enumeration becomes intractable. Since we are interested in computing the values of $\mathsf{E}^{\mathcal{D}}$ for large n and moderate t, we require a different strategy. Fortunately, solution is straightforward.

The weight distribution of the input set $\{0,1\}^k$ is a vector of rational numbers $\delta \in \mathbb{Q}^{[0,k]}$ and is easy to compute. For $w \in [0,k]$, there are $\delta_w := \binom{k}{w}$ inputs with Hamming weight w. The output weight distribution $\delta' \in \mathbb{Q}^{[0,n]}$ for an enumerator $\mathsf{E}^{\mathcal{D}}$ can then be computed as

$$\delta'_h = \sum_{w=0}^{k} \frac{\delta_w}{\binom{k}{w}} \mathsf{E}^{\mathcal{D}}_{w,h}$$

for $h \in [0,n]$. Here, one can think of $\frac{\delta_w}{\binom{k}{w}}$ as a fraction denoting the probability that any given input string is present in the input. Given that δ corresponds to the set of all input strings, we have $\frac{\delta_w}{\binom{k}{w}} = 1$. However, this observation can be used to compute the final weight distribution iteratively. Concretely, if $\mathbf{G} := \mathbf{G}_1\pi_1\mathbf{G}_2 \ldots \pi_{t-1}\mathbf{G}_t$, then we can initialize $\delta_{1,w} := \binom{k}{w}$ and compute

$$\delta_{i+1,h} := \sum_{w=0}^{n_i} \frac{\delta_{i,w}}{\binom{n_i}{w}} \mathsf{E}^{\mathcal{D}_i}_{w,h}.$$

for $i \in [t-1]$ and $h \in [0, n_{i+1}]$. The final weight distribution is therefore δ_t. Again, the fraction $\frac{\delta_{i,w}}{\binom{n_i}{w}}$ can be thought of as the probability that any given length-n_i input string is currently in the weight distribution. In summary, approach results in a total of $O((t-1)n^2)$ iterations instead of the $O(n^t)$ iterations and is purely a better way to group "common terms."

5.4 Systematic Enumerator

We consider a special type of parallel concatenation of the identity matrix $\mathbf{I} \in \{0,1\}^{k \times k}$ and an arbitrary $\mathbf{G}_1 \leftarrow \mathcal{D}_1$. Let $\mathbf{G} := (\mathbf{I}||\mathbf{G}_1) \in \{0,1\}^{k \times n}$ be the concatenated code with distribution $\mathcal{D}$. Then, the enumerator is

$$\mathsf{E}^{\mathcal{D}}_{w,h} = \mathsf{E}^{\mathcal{D}_1}_{w,h-w}.$$

Since $\mathbf{I}$ will contribute exact weight w to the output, the overall enumerator is just the number of ways to map weight w to $h-w$ via $\mathbf{G}_1 \in \{0,1\}^{k \times n-k}$.

Iterative Enumeration. At times, we will be interested in the case of $\mathbf{G}_1 := \mathbf{G}_{1,1}\pi_1\mathbf{G}_{1,2}...\pi_{t-1}\mathbf{G}_{1,t}$ being a serially concatenated code, and therefore

$$\mathsf{E}^{\mathcal{D}_1}_{w,h} = \sum_{h_1=0}^{n_1} ... \sum_{h_{t-1}=0}^{n_{t-1}} \frac{\mathsf{E}^{\mathcal{D}_{1,1}}_{w,h_1}...\mathsf{E}^{\mathcal{D}_{1,t}}_{h_t,h}}{\binom{n_1}{h_1}...\binom{n_{t-1}}{h_{t-1}}}$$

where $w \in [0,k]$ and $h \in [0, n-k]$. The systematic enumerator can then be expressed as

$$\mathsf{E}^{\mathcal{D}}_{w,h} = \sum_{h_1=0}^{n_1} ... \sum_{h_{t-1}=0}^{n_{t-1}} \frac{\mathsf{E}^{\mathcal{D}_{1,1}}_{w,h_1}...\mathsf{E}^{\mathcal{D}_{1,t}}_{h_t,h-w}}{\binom{n_1}{h_1}...\binom{n_{t-1}}{h_{t-1}}}$$

and evaluated iteratively as before, where the last iteration uses $h' := h - w$ in place of h.

Unfortunately, this prevents iterative enumeration as described above, since the input weight must be known when enumerating the final subcode. The limitation can be partially resolved by first building $\mathsf{E}^{\mathcal{D}}_{w,h}$ in an iterative way by first combining $\mathsf{E}^{\mathcal{D}_{1,1}}$ and $\mathsf{E}^{\mathcal{D}_{1,2}}$ before combining that enumerator with $\mathsf{E}^{\mathcal{D}_{1,3}}$, etc. Compared to the iterative enumeration above, this approach requires $O(n)\times$ more work, which significantly limits the parameters that can be considered.

6 Block-Diagonal Subcode

This section studies Block-Diagonal subcodes in isolation to expose their structure and limitations. These subcodes will serve as the outer layer of our final Block-Accumulate construction in Section 7 and serial Block-Diagonal codes in Section 8. The generator matrix $\mathbf{G} \in \{0,1\}^{k \times n}$ of the subcode can be written

in Block-Diagonal form; i.e., it can be decomposed into submatrices (which we refer to as blocks) $\mathbf{G}_i \in \{0,1\}^{\sigma \times e\sigma}$, $i \in [k/\sigma]$, placed along the diagonal:

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_1 & \mathbf{O} & \dots & \mathbf{O} \\ \mathbf{O} & \mathbf{G}_2 & \dots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \dots & \mathbf{G}_{k/\sigma} \end{pmatrix},$$

where $\mathbf{O}$ is the $\sigma \times e\sigma$ zero matrix and $e := \frac{n}{k}$ is the expansion ratio. Our instantiations will only consider $e \in \{1,2\}$, but other values can also be used. In this subsection, $\mathcal{D}$ denotes the uniform Block-Diagonal distribution where each $\mathbf{G}_i$ is sampled independently and uniformly from $\{0,1\}^{\sigma \times e\sigma}$; it imposes no per-block minimum-distance condition.

6.1 Block-Diagonal Enumerator

Let $\mathbf{x} \in \{0,1\}^k$ be an input word and denote the subwords as

$$\mathbf{x} = \left(\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_{k/\sigma}\right),$$

where $\mathbf{x}_i \in \{0,1\}^\sigma$, $i \in [k/\sigma]$. Then, the codeword $\mathbf{c} = \mathbf{x}\mathbf{G}$ corresponding to $\mathbf{x}$ is given by $\mathbf{c} = \left(\mathbf{c}_1 \ \mathbf{c}_2 \ \dots \ \mathbf{c}_{k/\sigma}\right)$, where $\mathbf{c}_i = \mathbf{x}_i\mathbf{G}_i \in \{0,1\}^{e\sigma}$, $i \in [k/\sigma]$.

Our goal is to construct the enumerator $\mathsf{E}^{\mathcal{D}}_{w,h}$, which counts the number of input-output pairs $(\mathbf{x}, \mathbf{y})$, such that $|\mathbf{x}|_{\mathsf{H}} = w$ and $\mathbf{y} := \mathbf{x}\mathbf{G}$ with $|\mathbf{y}|_{\mathsf{H}} = h$. However, given that $\mathbf{G} \leftarrow \mathcal{D}$ is essentially a random variable, there will not be a concrete number of such pairs. Moreover, for an arbitrary $\mathbf{G}$, it appears intractable to compute the enumerator. Instead, as discussed in Section 5.2, we will compute the expectation of the enumerator for the random variable $\mathbf{G} \leftarrow \mathcal{D}$.

We will achieve this in two phases by decomposing the main enumerator $\mathsf{E}^{\mathcal{D}}$ into two subenumerators for related properties. Given a weight w input $\mathbf{x}$, we will count the number of ways it could result in q subwords $\mathbf{x}_i$ being non-zero. This is a type of enumerator except that it does not count the binary Hamming weight, but the Hamming weight with elements in $\{0,1\}^\sigma$. One can then construct a second enumerator that counts the number of ways to map q non-zero subwords to and output word $\mathbf{c}$ with Hamming weight h.

In more detail, let us define $\mathsf{Q} : \{0,1\}^k \to 2^{[k/\sigma]}$, where $\mathsf{Q}(\mathbf{x}) \subseteq [k/\sigma]$ is the set of i such that $\mathbf{x}_i$ is non-zero. Additionally, define $\mathsf{q} : \{0,1\}^k \to [0, k/\sigma]$, where $\mathsf{q}(\mathbf{x}) = |\mathsf{Q}(\mathbf{x})|$. We slightly abuse notation by also defining $\mathsf{Q} : \{0,1\}^n \to 2^{[k/\sigma]}$, where $\mathsf{Q}(\mathbf{c}) \subseteq [k/\sigma]$ is the set of i such that $\mathbf{c}_i$ is non-zero. We now have the following lemma.

Lemma 1. *For $\mathbf{G} \leftarrow \mathcal{D}$ and any $\mathbf{x} \in \{0,1\}^k$, $\mathbf{y} \in \{0,1\}^n$,*

$$\Pr\left[X_{\mathbf{x},\mathbf{G},\mathbf{y}} = 1\right] = \begin{cases} 2^{-e\sigma \cdot \mathsf{q}(\mathbf{x})} & \text{if } \mathsf{Q}(\mathbf{y}) \subseteq \mathsf{Q}(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases}.$$

Proof. Clearly, if $\mathbf{x}_i = 0^\sigma$, then $\mathbf{c}_i = 0^{e\sigma}$. Thus, $\mathsf{Q}(\mathbf{c}) \subseteq \mathsf{Q}(\mathbf{x})$. In other words, if $\mathsf{Q}(\mathbf{y}) \not\subseteq \mathsf{Q}(\mathbf{x})$, then $\Pr[X_{\mathbf{x},\mathbf{G},\mathbf{y}} = 1] = 0$.

Consider any $\mathbf{x} \in \{0,1\}^k$ and $\mathbf{y} \in \{0,1\}^n$ such that $\mathsf{Q}(\mathbf{y}) \subseteq \mathsf{Q}(\mathbf{x})$. For all $i \in [k/\sigma]$, let $\mathbf{G}_i = \left(\mathbf{G}_{i,1}\ \mathbf{G}_{i,2} \dots \mathbf{G}_{i,e\sigma}\right)$, where $\mathbf{G}_{i,j} \in \{0,1\}^\sigma$ are the columns of $\mathbf{G}_i$ for $j \in [e\sigma]$. Let $\mathbf{y} = \left(\mathbf{y}_1\ \mathbf{y}_2 \dots \mathbf{y}_{k/\sigma}\right)$, where $\mathbf{y}_i \in \{0,1\}^{e\sigma}$, $i \in [k/\sigma]$, and let $\mathbf{y}_i = \left(\mathbf{y}_{i,1}\ \mathbf{y}_{i,2} \dots \mathbf{y}_{i,e\sigma}\right)$, where $\mathbf{y}_{i,j} \in \{0,1\}$ are the bits of $\mathbf{y}_i$ for $j \in [e\sigma]$. Consider $i \in \mathsf{Q}(\mathbf{x})$, i.e., $\mathbf{x}_i \neq 0^\sigma$. Then, for $j \in [e\sigma]$, $\Pr[\mathbf{y}_{i,j} = \mathbf{x}_i\mathbf{G}_{i,j}] = \frac{1}{2}$. Therefore, $\Pr[\forall i \in \mathsf{Q}(\mathbf{x}) : \mathbf{y}_i = \mathbf{x}_i\mathbf{G}_i] = 2^{-e\sigma\cdot\mathsf{q}(\mathbf{x})}$. Since $\mathsf{Q}(\mathbf{y}) \subseteq \mathsf{Q}(\mathbf{x})$, $\Pr[\forall i \notin \mathsf{Q}(\mathbf{x}) : \mathbf{y}_i = \mathbf{x}_i\mathbf{G}_i] = 1$. Therefore,

$$\Pr[X_{\mathbf{x},\mathbf{G},\mathbf{y}} = 1] = 2^{-e\sigma\cdot\mathsf{q}(\mathbf{x})}.$$

□

We now have

$$\mathbb{E}_{\mathbf{G}\leftarrow\mathcal{D}}\left[E^{\mathbf{G}}_{w,h}\right] = \sum_{\substack{\mathbf{x}\in\{0,1\}^k,\mathbf{y}\in\{0,1\}^n \\ \mathsf{Q}(\mathbf{y})\subseteq\mathsf{Q}(\mathbf{x}) \\ |\mathbf{x}|_\mathsf{H}=w,|\mathbf{y}|_\mathsf{H}=h}} 2^{-e\sigma\cdot\mathsf{q}(\mathbf{x})},$$

i.e., we need to count the number of $\mathbf{x} \in \{0,1\}^k$, $\mathbf{y} \in \{0,1\}^n$ with $\mathsf{Q}(\mathbf{y}) \subseteq \mathsf{Q}(\mathbf{x})$, $|\mathbf{x}|_\mathsf{H} = w$, and $|\mathbf{y}|_\mathsf{H} = h$. We perform this count using the intermediate variable $q = \mathsf{q}(\mathbf{x})$. We first compute the number of $\mathbf{x} \in \{0,1\}^k$ such that $|\mathbf{x}|_\mathsf{H} = w$ and $\mathsf{q}(\mathbf{x}) = q$. This is equal to the number of $\mathbf{x} \in \{0,1\}^k$ such that $|\mathbf{x}|_\mathsf{H} = w$ and $\mathsf{Q}(\mathbf{x}) = Q$ summed over all $Q \subseteq [k/\sigma]$ with $|Q| = q$. For any fixed $Q \subseteq [k/\sigma]$ with $|Q| = q$, the number of $\mathbf{x} \in \{0,1\}^k$ such that $|\mathbf{x}|_\mathsf{H} = w$ and $\mathsf{Q}(\mathbf{x}) = Q$ is equivalent to the number of ways in which we can assign w identical balls into q labeled slotted bins each having a capacity of σ such that none of the bins are empty, which is $S(w,q,\sigma)$. The number of $Q \subseteq [k/\sigma]$ with $|Q| = q$ is $\binom{k/\sigma}{q}$. Therefore, the number of $\mathbf{x} \in \{0,1\}^k$ such that $|\mathbf{x}|_\mathsf{H} = w$ and $\mathsf{q}(\mathbf{x}) = q$ is

$$\mathsf{E}_{w,q} := \binom{k/\sigma}{q} \cdot S(w,q,\sigma) = \binom{k/\sigma}{q} \cdot \sum_{i=0}^{q} (-1)^i \binom{q}{i}\binom{\sigma(q-i)}{w}.$$

Now, consider any $\mathbf{x} \in \{0,1\}^k$ such that $|\mathbf{x}|_\mathsf{H} = w$ and $\mathsf{q}(\mathbf{x}) = q$. We will count the number of $\mathbf{y} \in \{0,1\}^n$ such that $|\mathbf{y}|_\mathsf{H} = h$ and $\mathsf{Q}(\mathbf{y}) \subseteq \mathsf{Q}(\mathbf{x})$. This is simply equal to the number of ways of choosing h bits among the $e\sigma q$ bits of $\mathbf{y}_i$ for $i \in \mathsf{Q}(\mathbf{x})$ to set to 1. The number of ways to do this is thus $E_{q,h} := \binom{e\sigma q}{h}$. Putting everything together, we have

$$\begin{aligned}\mathbb{E}_{\mathbf{G}\leftarrow\mathcal{D}}\left[E^{\mathbf{G}}_{w,h}\right] &= \sum_{\substack{\mathbf{x}\in\{0,1\}^k,\mathbf{y}\in\{0,1\}^n \\ \mathsf{Q}(\mathbf{y})\subseteq\mathsf{Q}(\mathbf{x}) \\ |\mathbf{x}|_\mathsf{H}=w,|\mathbf{y}|_\mathsf{H}=h}} 2^{-e\sigma\cdot\mathsf{q}(\mathbf{x})} = \sum_{q=0}^{k/\sigma} 2^{-e\sigma q} \cdot E_{w,q} \cdot E_{q,h} \\ &= \sum_{q=0}^{k/\sigma} \left[2^{-e\sigma q} \cdot \binom{k/\sigma}{q}\binom{e\sigma q}{h} \cdot \sum_{i=0}^{q} (-1)^i \binom{q}{i}\binom{\sigma(q-i)}{w}\right].\end{aligned}$$

Optimization. We remark that it is possible to precompute some of the terms used across the iterations. The scaling factor $\frac{D_w^i}{\binom{k}{w}}$ is only a function of w but appears in kn different terms. Therefore one can cache these values. Similarly, one can precompute $v_q = 2^{-e\sigma q} \cdot \binom{k/\sigma}{q}$ for each value of $q \in [0, n/\sigma]$. The distribution is then computed as:

$$F_w := \frac{D_w}{\binom{k}{w}}, v_q := 2^{-e\sigma q} \cdot \binom{k/\sigma}{q}, S(w,q,\sigma) = c_{q,w} := v_q \cdot \sum_{i=0}^{q} (-1)^i \binom{q}{i} \binom{\sigma(q-i)}{w},$$

$$E_{w,h} := \sum_q v_q c_{q,w} \binom{e\sigma q}{h}, \delta_h := \sum_w F_w \cdot \sum_q c_{q,w} \binom{e\sigma q}{h}$$

Systematic Ensembles. An optimized version of our subcode instantiates $\mathbf{G}$ in systematic form: $\mathbf{G} := (\mathbf{I} \parallel \mathbf{G}')$. Here, $\mathbf{G}' \in \{0,1\}^{k \times (n-k)}$ is a Block-Diagonal matrix as defined in Section 7 and 8. This provides two primary benefits:

- Computational Efficiency: For a rate 1/2 code, the cost of the first matrix multiplication is reduced by 2×.
- Guaranteed Full Rank: Defining $\mathbf{G}$ in systematic form ensures it always has full row rank, which prevents the distance from collapsing to 0.

6.2 Empirical Weight Distribution

In this section, we examine the weight distribution of a non-systematic single Block-Diagonal matrix and the effect σ has on it. We present the enumerator as a contour plot in Figure 1 along with the plot of the final distribution of codewords δ_h.

The z-axis of the contour plot is the $\log_2$ expected number of weight-w inputs mapping to weight-h outputs. We place a colored gradient on the contour plot: green indicates that fewer than one codeword is expected, yellow indicates approximately one expected codeword, and red denotes that many codewords are expected. Unsurprisingly, this code does not achieve linear minimum distance as there must exist codewords with distance at most σ. This can easily be seen from the fact that the lower scatter plot does not go below $\log \delta_h = 0$ in Figure 1 near $h = 0$. However, we believe that it is instructive to examine the distribution before generalizing to $t > 1$.

Compared to a random linear code of the same size, the enumerator is very similar except in the low-weight regime where w, h are small. In this regime, we observe a large skewing of the distribution towards the origin. Moreover, one can also observe a rippling pattern in the distribution. This is apparent when inspecting the $\mathsf{E}_{w,h} = 0$ and $\mathsf{E}_{w,h} = 10$ contour lines near the origin. These ripples are due to the distribution of when a single input chunk $\mathbf{x}_i \in (\mathbf{x}_1, ..., \mathbf{x}_{k/\sigma})$ has a non-zero value. Among inputs supported on a single nonzero chunk, weight $w = \sigma/2$ gives the largest number of input words, and each such input maps to a uniformly distributed 2σ-bit codeword. Therefore, for such values of w, the large number of corresponding input words makes it likely that some map to

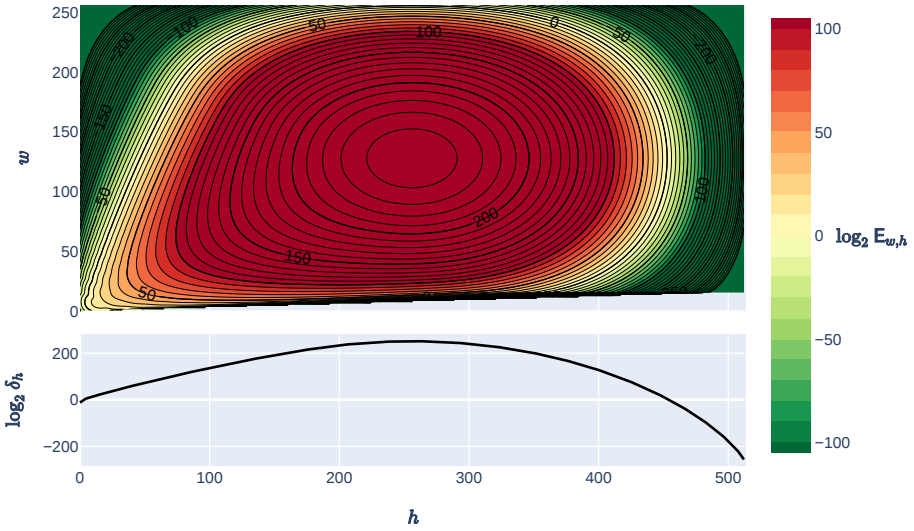


Fig. 1. Depictions of the Block-Diagonal enumerator $\mathsf{E}^{\mathcal{D}}$ with $(t, k, n, \sigma) = (1, 256, 512, 16)$. The x-axis is the output weight $h \in [0, n]$, y-axis of the plot is the input weight $w \in [0, k]$ and the z value of the contour plot is the value $\log_2(\mathsf{E}_{w,h})$. The bottom plot has the output distribution $\delta_h = \sum_{w=0}^{k} \mathsf{E}_{w,h}$.

codewords of small output weight h. This phenomenon repeats when two, three, or more chunks of $\mathbf{x}$ have non-zero values.

When σ becomes sufficiently small, e.g. $\sigma \leq 4$, we observe that the code begins to break down. We believe this is primarily due to the high probability of sampling a block that is not invertible, e.g., when a row of $\mathbf{G}_i$ is zero or contains duplicate rows. Figure 2 contains the distribution for the case of $\sigma = 2$. Indeed, this becomes apparent because all weights up to approximately $w \leq 60$ have at least one output with expected weight 0. To mitigate this, one could restrict the distribution to only sampling invertible matrices. However, this complicates the sampling algorithm and will not be needed in our final constructions. We leave investigating this for future work.

Additionally, with such a small value of σ, an input of weight w has its output weight h bounded above by $we\sigma$. As seen in Figure 2, this essentially squashes the higher weight codewords into smaller values of h, hurting the overall distance.

7 Block-Accumulate Codes

We explore the possibility of combining Block-Diagonal subcodes with Accumulators. While Block-Diagonal codes alone do not achieve linear minimum distance, the addition of accumulator layers strongly suppresses the low-weight tail, thereby dramatically improving the minimum distance. Given the flexibility of

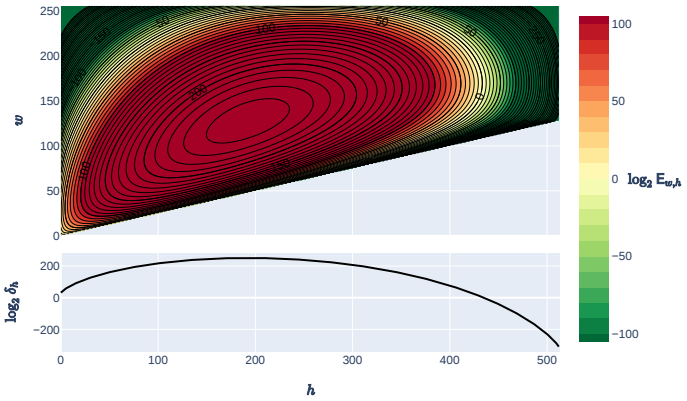


Fig. 2. Depictions of the Block-Diagonal enumerator $\mathsf{E}^{\mathcal{D}}$ with $(t, k, n, \sigma) = (1, 256, 512, 2)$. See Figure 1 for details.

our framework, combining different subcodes is straightforward, and this is the construction we recommend for practical PCGs and ZK systems.

The key additional ingredient is the accumulator, which is a simple recursive convolution. Although an accumulator has free distance one and therefore can be trivially "turned off" by structured inputs, its recursive structure causes even moderately sized inputs, once randomized by the BD subcode and interleavers, to expand into outputs of large Hamming weight. This amplification effect is precisely why accumulators play a central role in turbo and serial codes.

We tested variants that alternate between Block-Diagonal subcodes and Accumulators, as well as constructions consisting of a single Block-Diagonal subcode followed by $t - 1$ Accumulators. The latter exhibits significantly better minimum distance and is the code we focus on in this section. Specifically, we define Block-Accumulate (BA-t) codes with state size σ and depth t as

$$\mathbf{G} := \mathbf{B}\pi_1\mathbf{A} \cdots \pi_{t-1}\mathbf{A},$$

where $\mathbf{A} \in \{0,1\}^{n\times n}$ is an Accumulator and $\mathbf{B} \in \{0,1\}^{k\times n}$ is sampled from the systematic Block-Diagonal distribution with state size σ. We focus on $t = 3$, for which we show that the resulting code achieves linear minimum distance with strong concrete parameters.

7.1 Enumeration

Figure 3 contains the enumerator distribution for the case of $k = 256$ and $\sigma = 16$. With the exception of some low-weight inputs, we observe that the distribution is extremely close to that of a random code. In particular, we observe that the probability of codewords with substantially smaller weight than a random code is $\approx 2^{-\sigma}$. This is more clearly illustrated in Figure 4, which plots the final weight distribution for $\sigma \in \{4, 8, 16, 32\}$, along with Repeat-Accumulate code (RA-3) and a random linear code. As with Block-Diagonal codes, we again observe a phase transition where the probability of low-weight codewords decreases slowly

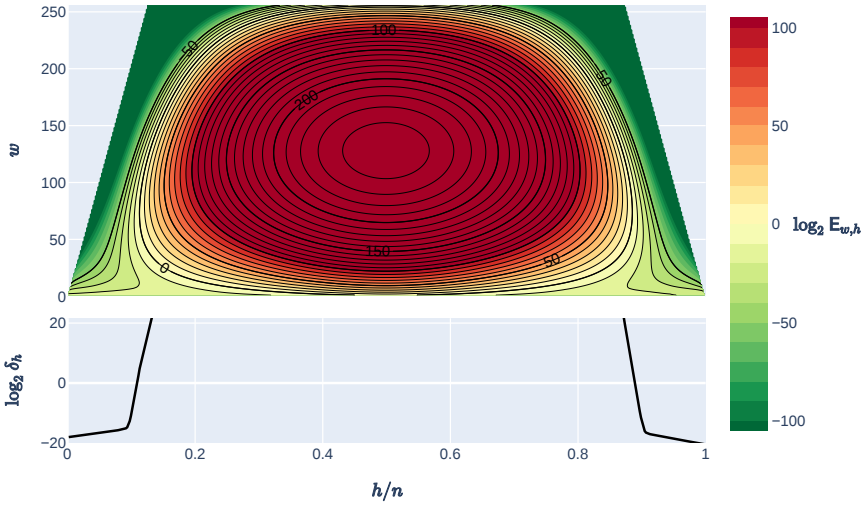


Fig. 3. Depictions of the Block-Accumulate enumerator $\mathsf{E}^{\mathcal{D}}$ with $(t, k, n, \sigma) = (3, 256, 512, 16)$. See Figure 1 for details.

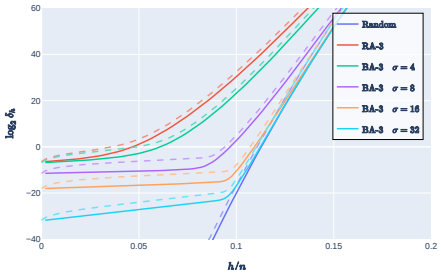


Fig. 4. Depictions of the Block-Accumulate log weight distribution. The dashed lines denote the cumulative log weight distribution.

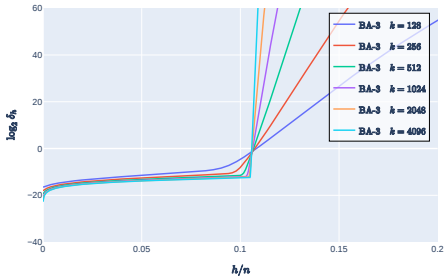


Fig. 5. Depictions of the Block-Accumulate cumulative log weight distribution for $\sigma = 16$ and various k.

for small values of h. However, this phase transition happens below $\log \delta_h = 0$, even for small values of σ.

Moreover, unlike Block-Diagonal codes, as k increases there is no need to increase σ. We instead see the opposite behavior in that the distance slightly improves as k increases. This can be seen in Figure 5, which plots the cumulative log weight distribution for a fixed $\sigma = 16$ and $k \in \{128, 256, 512, 1024, 2048, 4096\}$. We also observe that the phase transition becomes more acute and approximately occurs at $\log_2 \delta_h = -12$ and $h/n = .105$. We investigate various values of σ, k and conclude that the general trend is that the cumulative phase transition occurs at $-\frac{\sigma 3}{4}$ and relative minimum distance 0.105. This immediately gives a simple parameterization of setting $\sigma = 4/3\lambda$ for a guarantee that the relative minimum distance is 0.1 except with probability $2^{-\lambda}$.

7.2 Asymptotic Analysis of Block-Accumulate Codes

To prove the asymptotic behavior of our construction we follow the structure of [BMS09]. This proof is separate from the prior uniform enumerator ensemble: we fix $\mathbf{B}$ and assume each active block contributes at least a constant $d' > 4$. This condition can be enforced by rejection sampling, or by selecting a good subcode and repeating it across all blocks of $\mathbf{B}$. This analysis will demonstrate that the code will have linear minimum distance and we conjecture it will give concrete distance at least as good as the bounds obtained from the enumeration. The full proof is found in Appendix B and below is a summary of the core argument.

Proof Sketch (Linear Minimum Distance). Let $\mathbf{G} = \mathbf{B}\pi_1\mathbf{A}\pi_2\mathbf{A}$, where π_1, π_2 are uniform interleavers and $\mathbf{A}$ is the standard length-n accumulator. We show that for a sufficiently small constant $\varepsilon > 0$, the expected number $E_{\leq \varepsilon n}$ of nonzero codewords of weight $\leq \varepsilon n$ tends to zero, and conclude by Markov.

Fix a nonzero input $\mathbf{x}$ and define intermediate weights $u := |\mathbf{xB}|_{\mathsf{H}}$, $h_1 := |\mathbf{xB}\pi_1\mathbf{A}|_{\mathsf{H}}$, and $h := |\mathbf{xB}\pi_1\mathbf{A}\pi_2\mathbf{A}|_{\mathsf{H}}$. Random interleavers imply that conditioned on weight, the vector entering each accumulator stage is uniform over all vectors of that weight; thus each stage is governed by the accumulator input-output enumerator. The key analytic estimate is the accumulator small-output contraction (Lemma 2): for $t \leq 2\varepsilon n$,

$$\Pr[|z\mathbf{A}|_{\mathsf{H}} = t \mid |z|_{\mathsf{H}} = s] \leq \left(4e^2 \cdot \tfrac{t}{n}\right)^{s/2},$$

and summing over $1 \leq h \leq \varepsilon n$ gives

$$\Pr[h \leq \varepsilon n \mid h_1] \leq (\varepsilon n)\,(4e^2\varepsilon)^{h_1/2}.$$

Thus achieving $h \leq \varepsilon n$ forces h_1 to be small, and is strongly suppressed in h_1.

We now exploit the block structure of $\mathbf{B}$. Let $b = b(\mathbf{x})$ be the number of active σ-bit input blocks. Assumption (1)implies $u = |\mathbf{xB}|_{\mathsf{H}} \geq d'b$, and the number of inputs with exactly b active blocks is at most $\binom{k/\sigma}{b}(2^\sigma - 1)^b$. Summing over inputs grouped by b, and using the first-accumulator contraction with $u \geq d'b$ together with the second-accumulator tail bound above, yields an upper bound of the form

$$E_{\leq \varepsilon n} \;\leq\; (\varepsilon n)\, C_0 \sum_{b=1}^{\lfloor 4\varepsilon n/d' \rfloor} \left[\, K(\sigma, R, d', \varepsilon) \cdot b^{d'/2-1} \cdot n^{\,1-d'/2} \right]^b,$$

for an explicit constant $K(\sigma, R, d', \varepsilon)$.

The bottleneck is the $b = 1$ term:

$$E_{b=1} \;\leq\; (\varepsilon n) \cdot O(n^{1-d'/2}) \;=\; O(n^{2-d'/2}),$$

so we require $2 - d'/2 < 0$, i.e., $d' > 4$. Under $d' > 4$ and sufficiently small constant ε, the remaining terms $b \geq 2$ are $o(1)$ even after the outer factor (εn):

for $2 \le b \le \sqrt{n}$ we have $(b/n)^{d'/2-1} \le n^{-(d'/2-1)/2}$ so the sum contributes $O(n^{-(d'/2-1)})$, while for $b \ge \sqrt{n}$ the truncated tail is geometrically small in b and hence $O(\rho^{\sqrt{n}})$ for some $\rho < 1$. Therefore $E_{\le \varepsilon n} \to 0$, and Markov gives $\Pr[d_{\min}(\mathbf{G}) \ge \varepsilon n] \to 1$. The full proof (including constant choices and all intermediate bounds) appears in Appendix B.

7.3 Discussion

The proof in Appendix B and the finite-n enumerator use two closely related, but not identical, block ensembles. The proof fixes $\mathbf{B}$ and assumes a per-block minimum-distance lower bound $d' > 4$; this conservative condition removes rare local bad blocks and makes the union bound clean. The enumerator instead uses the natural systematic-random block distribution, which is the distribution used to set concrete parameters. In practice, one can bridge the two by rejection sampling only the blocks that fail the local $d' > 4$ check, or by choosing one good local block code and reusing it. This can only remove very low local outputs, and in our enumerator experiments even an artificial repair that moves all mass below d' up to $d' + 1$ changes the predicted threshold only slightly.

We keep the two ensembles separate because they serve different purposes. The enumerator computation relies on the uniformity of each block to remain tractable: once an active input block is fixed, its output is uniformly distributed over the corresponding local output space, which we use in the counting formulas. The asymptotic proof, by contrast, needs a worst-case lower bound on how much weight any active block contributes before the interleavers and accumulators. A uniformly sampled small block can occasionally violate this condition, even though such blocks are rare and empirically harmless at our chosen parameters. Conditioning the block distribution on passing the local distance test removes this nuisance for the proof but destroys the exact uniform counting identity used by the enumerator.

For deployment, this distinction is conservative. If one only needs the asymptotic linear-distance statement, one can sample from the proof ensemble, e.g., by rejection sampling bad blocks or by selecting a good local block code and reusing it. This fixed-good-block option is especially natural: it removes the chance of locally weak blocks entirely while preserving the same global interleaver/accumulator mechanism. If one wants concrete parameters, the uniform enumerator gives a sharp prediction for the natural implementation, and rejecting bad local blocks is an additional safety measure that intuitively should improve local distance behavior. We do not claim a theorem that the conditioned ensemble has an enumerator no worse than the uniform ensemble; proving such monotonicity appears difficult because conditioning changes the block-level output distribution. Instead, our claim is narrower: the proof ensemble establishes the asymptotic mechanism, while the uniform enumerator gives the concrete parameter estimates used in the table.

Both analyses point to the same dominant failure mode: inputs that activate a single σ-bit block of $\mathbf{B}$. In the proof this is the $b = 1$ term, which is also why the condition $d' > 4$ is needed after accounting for the second-accumulator tail

bound. Inputs activating $b \geq 2$ blocks are suppressed much more strongly, since the accumulator contraction acts on weight at least $d'b$ and the suppression is raised with b. This explains the empirical "floor" in the low-weight tail: the number of single-block candidates is about $(k/\sigma)2^{\sigma}$, while the probability that any fixed one has anomalously low final weight is driven mainly by the interleavers and accumulators.

This single-block mechanism also explains why the finite-size curves stabilize in the regime we care about. Increasing n adds more candidate blocks, but it also gives the two random interleavers and accumulators more room to spread a fixed local contribution. Once σ is fixed, the low-weight tail is therefore governed by a competition between a growing number of local inputs and a shrinking probability that any fixed local input remains low weight after two accumulator stages. The plotted curves suggest that this competition quickly approaches a stable threshold near relative distance 0.1 for the parameters we use. The theorem is intentionally too coarse to prove this constant, but it supports the same qualitative picture: after the $b = 1$ term is controlled, the multi-block terms decay with substantially more slack.

Thus the asymptotic theorem should be read as proving linear minimum distance and identifying the low-weight mechanism, not as proving the exact limiting distance constant or monotonic finite-n behavior. The sharper concrete constants come from the enumerator. The smooth convergence of the enumerator curves, together with the proof's single-block mechanism, provides the evidence we use to extrapolate the concrete distance behavior to the practical lengths of interest.

8 Serially Concatenated Block-Diagonal Codes

Although the Block-Accumulate (BA-3) construction in Section 7 achieves stable relative minimum distance with a fixed state size σ, it is natural to ask what is achievable by pure serial, non-recursive architectures that do not use accumulators. In this section, we study the behavior of such constructions, focusing on block-diagonal serial codes as a theoretical baseline. This analysis clarifies the limitations of non-recursive designs and highlights why the BA-3 construction is able to surpass these bounds in the concrete regime.

8.1 Weight Distribution at $t = 2$

Now consider the case of t serially concatenated Block-Diagonal subcodes. Our high-level conclusion is that the code undergoes a phase transition at $\sigma = \Theta(\sqrt[t]{k})$, where for smaller values of σ, the code is expected to have sublinear minimum distance, while for larger values of σ, it has linear minimum distance in expectation. This result is intuitive given the structure of the code and it is also tight as smaller values of σ can be shown to have sublinear minimum distance.

Indeed, consider the case of $t = 2$ and $\sigma = O(\sqrt{k})$. It will be instructive to consider the case of what happens to a weight $w = 1$ input word. After the

first multiplication, the weight of the intermediate word is upper bounded by $\sigma = O(\sqrt{k})$. This intermediate vector will be randomly permuted. The number of "on" blocks for the second iteration will be upper bounded by σ. However, there are only $O(\sigma) = O(\sqrt{k})$ blocks, and therefore a constant fraction of the blocks will be on in expectation. As such, the resulting codeword should have linear Hamming weight. However, if $\sigma = o(\sqrt{k})$, this argument starts to break down. After the first iteration, less than a constant fraction of the $q = n/\sigma$ blocks will be on in expectation. Therefore, the resulting codeword must also have less than linear Hamming weight.

In Figure 6, we present the enumerator for $t = 2$ and $\sigma = 32$. Concretely, we find that as one varies k, setting $\sigma := 2\sqrt{k}$ results in the code achieving expected minimum distance approximately $0.1n = 0.2k$. This matches the same minimum distance as a random code of the same size. This can be observed in the lower plot of Figure 6, showing the $\log_2 \delta_h$ values for the various values of $h \in [0, 512]$. To more accurately depict the minimum distance, we have cropped the plot to only show values of $\log_2 \delta_h$ in the range $[-50, 50]$.

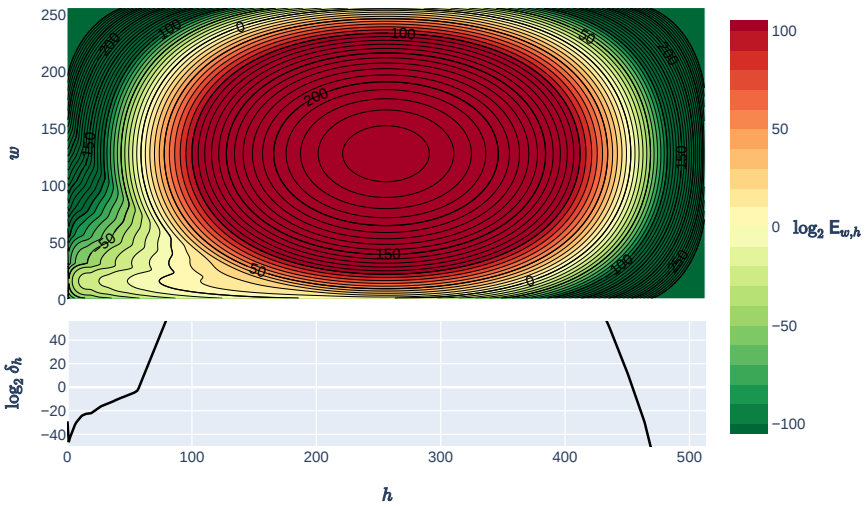


Fig. 6. Depictions of the Block-Diagonal enumerator $\mathsf{E}^{\mathcal{D}}$ with $(t, k, n, \sigma) = (2, 256, 512, 32)$. See Figure 1 for details.

As can be observed, there is a phase transition of this graph at approximately $\log_2 \delta_h = 0$ and $h = 56$. As h approaches this phase transition from zero, we observe a slow growth rate of $\log_2 \delta_h$; beyond the transition, the growth becomes rapid. The weight distribution above this phase transition is identical to that of a random code and therefore the best that can be achieved. However, below the phase transition, there is a significant difference between the Block-Diagonal codes and random codes. For random codes, we observe the expected number of codewords with a given h value to continue to decrease at a rapid rate. The exact location of the phase transition is a function of σ as seen in Figure 7.

For $\sigma = o(\sqrt{k})$, the phase transition happens above the $\log_2 \delta_h = 0$ line, and therefore, in expectation, we would see codewords with smaller h than a random code. For small values of σ, such as 2 and 4, the code degenerates without exhibiting a phase transition, and numerous low-weight codewords appear. We believe the lack of phase transition is largely due to having too many submatrices that are relatively far from being invertible. As such, when multiplying by such a matrix, information is lost, resulting in poor distance. We suspect that if one was able to define the enumerator for Block-Diagonal codes with only invertible submatrices, the code would behave better in this regime.

As σ increases beyond the phase transition $2\sqrt{k}$, we observe an exponential drop in the expected number of low-weight codewords as compared to a random code. For example, at $\sigma = 64$ there is less than a 2^{-30} probability of having a codeword with small weight compared to a random code.

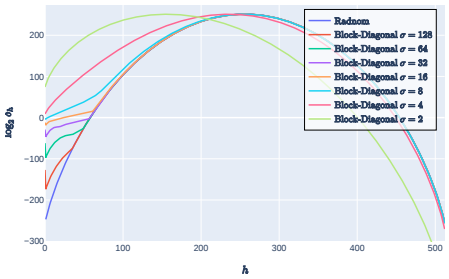


Fig. 7. Depictions of the Block-Diagonal weight distribution with $(t, k, n) = (2, 256, 512)$, various $\sigma \in \{2, 4, 8, 16, 32, 64, 125\}$ and a random linear code.

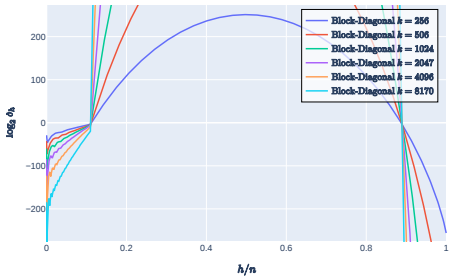


Fig. 8. Depictions of the Block-Diagonal weight distribution with various $k \in \{128, 506, 1025, 2047, 4096, 8170\}, n = 2k, \sigma = 2\sqrt{k}$.

Thus far, we have focused on the case of $k = 256$. However, we are also interested in general values of k such as $k \in [128, 2^{20}]$. In Figure 8, we plot the expected weight distribution for various values of k and $\sigma = 2\sqrt{k}$. The general trend is that for larger values of k, the probability of small weight codewords decreases. For example, at $k = 8170$ there is less than a 2^{-20} probability of being a codeword with $h \leq 0.11n$ and 2^{-100} probability of a codeword with $h \leq 0.05n$. While such bounds are more than sufficient for most applications, increasing σ further can reduce the probability of low-weight codewords, as shown in Figure 7.

8.2 Partially Systematic

We optimize the depth-t serial ensemble by making only the first subcode systematic: $\mathbf{G}_1 := (\mathbf{I} \parallel \mathbf{G}'_1)$, where $\mathbf{G}'_1 \in \{0,1\}^{k \times (n-k)}$ is Block-Diagonal. This reduces the first multiplication cost by $2\times$ at rate $1/2$ and ensures $\mathbf{G}_1$ has full row rank, preventing non-invertible diagonal blocks from collapsing the minimum distance. As shown in Figure 9, this partial systematization smooths the weight

distribution and significantly reduces the probability of low-weight codewords near $h = 0$ compared to the non-systematic version. While the phase transition point remains largely unchanged, the smoother contour lines suggest that avoiding non-invertible matrices at the first layer stabilizes the code's distance properties.

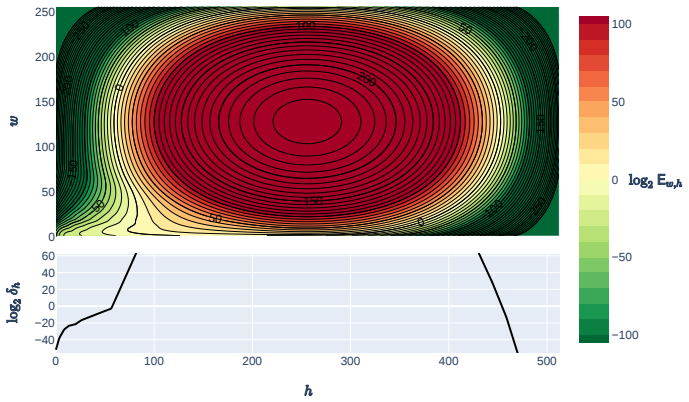


Fig. 9. Depictions of the systematic Block-Diagonal enumerator $\mathsf{E}^{\mathcal{D}}$ with $(t, k, n, \sigma) = (2, 256, 512, 32)$. See Figure 1 for details.

8.3 Linear Minimum Distance of Pure Block-Diagonal Code

Theorem 1 (Linear minimum distance, all constant $t \geq 3$, rate $1/2$). *Fix any constant $t \geq 3$ and rate $R = 1/2$. There exist constants $c > 0$, $\delta > 0$, and $\gamma > 0$ such that the following holds. Let $\sigma := \lceil c\, n^{1/t} \rceil$ rounded so that $\sigma \mid n$ and $\sigma \mid k = n/2$, and sample $\mathbf{G}$ from the pure block-diagonal serial ensemble above with independent uniform block matrices and independent uniform interleavers. Then for all sufficiently large n,*

$$\Pr\big[d_{\min}(\mathbf{G}) \leq \delta n\big] \;\leq\; \exp(-\gamma\sigma^2).$$

See Appendix F for the full proof.

9 Banded-Diagonal Codes

We consider an alternative formulation of our Block-Diagonal code, called the Banded-Diagonal code, where the diagonal submatrices are replaced by a diagonal band of random entries of width σ. We expect similar asymptotic performance, but conceptually this formulation offers some advantages. See Appendix G for details. In the Block-Diagonal case, 1s in the input $\mathbf{x}$ that fall within the same block do not change the weight distribution of the output $\mathbf{xG}$. That is, once a block is "on", additional 1s in the input $\mathbf{x}$ of this block do not

change the Hamming weight, since the output distribution of this block is already uniform. The idea of the banded construction is to offset each row so that there is some non-overlapping portion. That is, the generator of this subcode can be described as

$$\mathbf{G} = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \dots & \alpha_{1,\sigma} & & & \mathbf{O} \\ & \alpha_{2,1} & \alpha_{2,2} & \dots & \alpha_{2,\sigma} & & \\ & & \ddots & \ddots & & \ddots & \\ \mathbf{O} & & & \alpha_{k,1} & \alpha_{k,2} & \dots & \alpha_{k,\sigma} \end{pmatrix}.$$

More generally, each row will be offset from the previous by $(n-\sigma)/k$, which we assume to be an integer. We present the enumerator for this code in Appendix G. The computational cost of computing the enumerator proved to be higher than that of the Block-Diagonal codes, and therefore we chose not to implement it. However, we believe it could offer better concrete performance. Indeed, it is not hard to show that it too must have $\sigma \geq O(\sqrt[t]{k})$. We leave further investigation of this approach to future work. See Appendix G for more details.

10 Experimental Evaluation

We present concrete experimental results from our implementation. Section 10.1 reports general code performance, Section 10.2 evaluates its use in PCGs, and Section 10.3 covers its use in zero-knowledge protocols.

Implementation. We implement our framework on both CPU and GPU. Its modular design enables easy integration of various code architectures (e.g., BA-t, RA-t, Expand-Accumulate codes [BCG+22], etc.). Our implementation is incorporated into the `libOTe` C++ library [RR] as well as a fork of the Rust library `Plonkish` [ZH]. On the GPU, we use custom CUDA kernels alongside the Thrust C++ parallel algorithms library. For the permutation step, we rely on Thrust's `shuffle()`, which employs a Feistel-based bijection. For the Block-Diagonal code, we provide a dedicated CUDA kernel that assigns each block column to a separate thread. Since our initial goal was to use these codes for improved PCGs, we compress by $\mathbf{G}$ (from n to k) rather than expand by it (from k to n). Although $\mathbf{G}$ operates in $\mathbb{F}_2$, for PCG purposes we compress a vector of 128-bit elements by $\mathbf{G}$. For the PCG interactive setup, we use the GPU-optimized AES implementation described in [Tez21]. The code will be open-sourced.

Experimental Setting. We conduct our experiments on a commodity laptop, equipped with an upper mid-range 13th Gen Intel Core i7-13700H CPU running at 2.40 GHz, and with 32 GB of RAM. The laptop features a consumer-grade NVIDIA GeForce RTX 4060 GPU with 3072 CUDA cores and 8 GB of GDDR6.

Codes and Parameters. We present our recommended code parameters in Figure 10 along with their running times and compare them with the state of the art.

As discussed before, our most promising code is the Block-Accumulate-t code with $t = 3$ iterations, i.e. **BA-**3. We also present our Block-Diagonal code with $t = 2$ and $t = 3$ iterations (**BD-**t). We compare them to Expand-Accumulate (**EA**) [BCG+22] and Expand-Convolute (**EC**) [RRT23]. These codes have distance bounds either via Markov-chain analysis or by conjecture; the analytic bounds are believed to be loose, so prior works [BCG+22,RRT23] also propose plausibly tight conjectured parameters. We do not implement **BD** codes on the CPU, as their computational cost makes them unlikely to be competitive on this platform. We also do not implement **EC** codes on the GPU due to the need to implement iterative kernels that perform a two-pass convolution. See below for discussion.

Figure 10 includes the minimum-distance bounds supplied by each analysis. We report the relative minimum distance d, where the minimum distance is dn, and the one-time sampling failure level λ, where the code achieves distance dn with probability at least $1 - 2^{-\lambda}$. This is not a per-use failure event, and LPN/Syndrome Decoding does not immediately become insecure if a lower-distance code is sampled; an attacker must still find the low-weight codewords. We therefore use modest λ values: our aggressive parameters use $\lambda = 4$ with $\sigma = 8$, while our conservative parameters use $\lambda = 20$ with $\sigma = 32$ and distance 0.11. If the required distance for $\sigma = 32$ is decreased to 0.03, the same computation gives approximately $\lambda = 28$ bits of security. Prior works [BCG+22,RRT23] have also used aggressive parameters without distance bounds. In Figure 10, we mark such conjectured-distance parameter sets with a *.

10.1 Code Performance

We present the results of our experiments in Figure 10. We show that on both the CPU and GPU, our **BA-**3 code achieves the best performance. On the CPU, **BA-**3 improves over state-of-the-art **EC** [RRT23] $\approx 8\times$ (or $\approx 3\times$ when **EC** and our code use aggressive parameters). On the GPU, **BA-**3 improves over **EA** $\approx 50\times$ ($\approx 20\times$ with more conservative parameters).

10.2 Pseudorandom Correlation Generators

As described in Appendix A, a complete PCG protocol consists of (1) an interactive setup phase that yields the initial correlation, and (2) a local encoding by **G**. Both steps incur nontrivial cost, though the encoding by **G** is typically the dominant factor. This is particularly true when applying the recent Stationary Syndrome Decoding optimization of [KPRR25], where the setup is performed once and subsequent vectors are derived via local PRG calls. Our implementation shows that the one-time cost of generating the GGM tree is approximately 10 ms (amortized across many invocations), with each subsequent expand costing about 2 ms on the GPU (8 ms on the CPU), consisting of n calls to AES. In the GPU-optimized interactive setup, over 97% of runtime is spent in GPU-accelerated AES evaluations [Tez21]. While [Tez21] reports throughput of 100 GB/s, our laptop achieved only about 6 GB/s, suggesting that the 2 ms expand

Code		Min. Dist.	Bit Sec.	CPU (ms)	GPU (ms)
[This] BA-3	$\sigma = 8$	0.10	4	**21.4**	**3.2**
[This] BA-3	$\sigma = 32$	0.10	20	30.2	3.3
[This] BD-3	$\sigma = 256$	0.05	10	–	14.3
[This] BD-2	$\sigma = 2048$	0.05	10	–	40.0
[BCG+22] EA*	$w = 36$	0.05	10	598.0	73.0
[BCG+22] EA	$w = 78$	0.05	20	1,211.0	156.0
[RRT23] EC*	$w = 7, m = 25$	–	–	67.1	–
[RRT23] EC	$w = 33, m = 25$	0.05	20	235.6	–

Fig. 10. Comparison of our codes, constructed via various concatenations of Block-Diagonal codes and Accumulators, with state-of-the-art Expand-Accumulate (**EA**) [BCG+22] and Expand-Convolute (**EC**) [RRT23] codes on both CPU and GPU, for dimension $k = 2^{20}$. Reported times are for the linear-code encoding step only. Some uncompetitive configurations are omitted and marked with –. The Block-Diagonal code **B** is parameterized by state size σ, the Expander **E** is parameterized by column Hamming weight w, and the Convolution **C** is parameterized by memory m. Parameter sets whose distance is conjectured rather than bounded by the cited analysis are marked with a *.

cost could be reduced significantly on higher-end hardware. In our benchmarks, encoding remains the main bottleneck, taking roughly 3 ms compared to 2 ms for AES (versus 21ms and 8ms on the CPU). Overall, this translates to a projected[6] throughput of about 200+ million OTs per second on the GPU, excluding the one-time 10 ms GGM seed expansion. Since generating a binary Beaver triple consumes two such OT-style correlations, the corresponding throughput is about 100 million binary Beaver triples per second.

10.3 Integration: BA vs. RAA in Blaze

To evaluate the impact of our new codes, we integrated BA-3 into the binary zero knowledge Blaze system [BCF+25]. We modified the `plonkish_basefold` repository [ZH] to replace the RAA encoder with our BA-3 code. For an overview of the Blaze system, see Appendix A. We compared our BA-3 (rate 1/2) construction against the baseline RAA (rate 1/4) implementation for a polynomial of size $k = 2^{20}$. The results demonstrate significant improvements in both time and memory. Notably, because BA-3 provides a better native rate, we achieve a 2x speedup in the encoding phase and a halving of total memory usage. We note that [BCF+25] suggests using puncturing to reduce the rate to 1/2, however, this does not speed up or reduce the memory of the encoding/checking procedure, only compresses the codeword after encoding.

While the total proof size is slightly larger (due to the increased query count of 800 in the BA-3 test to maintain the same security margins), the internal

[6] We only implemented the linear code portion of the computation and performed PRG (AES) micro benchmark estimates on the cost of generating the error vectors via the GGM tree. The linear code time is expected to dominate and results in 300 million OTs/s.

Metric	RAA	BA-3	Improvement
Encode	2,572	1,329	$\sim$1.93$\times$
Merkelize	599	319	$\sim$1.87$\times$
Batch Open	39,828	12,022	$\sim$3.31$\times$
Verify	164	102	$\sim$1.60$\times$
Proof Size	1.2	1.6	$\sim$0.75$\times$

Table 1. Performance comparison of RAA (rate 1/4, 400 queries) and BA-3 (rate 1/2, 800 queries). All times are in milliseconds (ms). Proof size is in megabytes (MB).

steps of the prover show dramatic gains. The Commit/Encoding time, which is the foundational cost of the commitment phase, is reduced by approximately 2x.

In our current benchmarks, "Open/Batch Open" is the largest cost ($\approx$ 12 s vs $\approx$ 1.3 s for encoding). However, this ratio is deceptive for future systems. If we apply PipFri-style optimizations to the opening phase, we can expect the opening time to drop significantly (potentially by 10x). In that future scenario, the 1.3s encoding time of BA-3 versus the 2.5s encoding time of RAA becomes a critical differentiator, representing a much larger percentage of the total proof generation latency. BA-3 provides these benefits natively, making it a more robust and efficient choice for scaling SNARKs to billions of constraints.

References

AFS05. D. Augot, M. Finiasz, and N. Sendrier. A family of fast syndrome based cryptographic hash functions. In *Mycrypt*, pages 64–83, 2005.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 2087–2104, 2017.

BBC+24. Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$OLE-based multiparty computation for boolean circuits. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VI*, volume 15489 of *LNCS*, pages 69–101. Springer, Singapore, December 2024.

BCF+25. Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D Rothblum, and Hadas Zeilberger. Blaze: Fast snarks from interleaved raa codes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 123–152. Springer, 2025.

BCG+19a. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

BCG+19b. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.

BCG+20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Cham, August 2020.

BCG+22. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Cham, August 2022.

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

BDMP98. Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara. Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding. *IEEE Transactions on Information Theory*, 44(3):909–926, 1998.

Ber68. Elwyn R. Berlekamp. *Algebraic coding theory*. McGraw-Hill series in systems science. McGraw-Hill, 1968.

BFK+24. Alexander R Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic snarks from expand-accumulate codes. In *Annual International Cryptology Conference*, pages 276–307. Springer, 2024.

BGT93. Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo codes. In *Proceedings of ICC '93 - IEEE International Conference on Communications*, pages 1064–1070, 1993.

BMMS26. Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, and Matan Shtepel. Query-optimal iopps for linear-time encodable codes. In *Advances in Cryptology – EUROCRYPT 2026*, volume 16547 of *Lecture Notes in Computer Science*, pages 95–120, Cham, 2026. Springer Nature Switzerland.

BMS09. Louay Bazzi, Mohammad Mahdian, and Daniel A. Spielman. The minimum distance of turbo-like codes. *IEEE Transactions on Information Theory*, 55(1):6–15, January 2009.

Bro19. Kevin Brown. Probability and combinatorics. https://www.mathpages.com/home/kmath337/kmath337.htm, 2019.

BSBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Annual international cryptology conference*, pages 701–732. Springer, 2019.

BSCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. Aurora: Transparent succinct arguments for r1cs. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 103–128. Springer, 2019.

CRR21. Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 502–534, Virtual Event, August 2021. Springer, Cham.

DJM98. Dariush Divsalar, Hui Jin, and Robert J McEliece. Coding theorems for "turbo-like" codes. In *Proceedings of the annual Allerton Conference on Communication control and Computing*, volume 36, pages 201–210. University Of Illinois, 1998.

Ds17. Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 523–535. ACM Press, October / November 2017.

GLS+23. Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. Brakedown: Linear-time and field-agnostic snarks for r1cs. In *Annual International Cryptology Conference*, pages 193–226. Springer, 2023.

GWC19. Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive, Paper 2019/953*, 2019.

HOSS18. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 3–33, August 2018.

KPRR25. Vladimir Kolesnikov, Stanislav Peceny, Srinivasan Raghuraman, and Peter Rindal. Stationary syndrome decoding for improved pcgs. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, volume 16000 of *Lecture Notes in Computer Science*, pages 284–317, Cham, 2025. Springer Nature Switzerland.

KU98. Nabil Kahale and Rüdiger Urbanke. On the minimum distance of parallel and serially concatenated codes. In *Proceedings. 1998 IEEE International Symposium on Information Theory (Cat. No. 98CH36252)*, page 31. IEEE, 1998.

LXYY25. Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. In *Advances in Cryptology – EUROCRYPT 2025*, Lecture Notes in Computer Science, pages 145–175, Cham, 2025. Springer Nature Switzerland.

LZ25. Tianyi Liu and Yupeng Zhang. Efficient snarks for boolean circuits via sumcheck over tower fields. *Cryptology ePrint Archive, Paper 2025/594*, 2025.

RGiA11. Eirik Rosnes and Alexandre Graell i Amat. On the analysis of weighted nonbinary repeat multiple-accumulate codes. *arXiv preprint arXiv:1101.5966*, 2011.

RR. Peter Rindal and Lawrence Roy. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

RR22. Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2505–2517. ACM Press, November 2022.

RRT23. Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 602–632. Springer, Cham, August 2023.

RS21. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Cham, October 2021.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.

SGRR19. Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

Tez21. Cihangir Tezcan. Optimization of advanced encryption standard on graphics processing units. *IEEE Access*, 9:67315–67326, 2021.

VY12. Branka Vucetic and Jinhong Yuan. *Turbo codes: principles and applications*, volume 559. Springer Science & Business Media, 2012.

WYKW21. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.

YWL+20. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

ZH. Hadas Zeilberger and Ting Han. Plonkish. https://github.com/hadasz/plonkish_basefold.git.

Supplementary Material

A Applications

A.1 Pseudorandom Correlation Generators

PCG Overview. We focus on PCGs based on Syndrome Decoding (SD) / Learning Parity with Noise (LPN) for generating OT and VOLE correlations of size k [BCG+19b]. These protocols begin with the parties performing an interactive, efficient setup to produce random vectors $\mathbf{a}', \mathbf{b}', \mathbf{c}' \in \mathbb{F}^n$ and a scalar $\Delta \in \mathbb{F}$ satisfying

$$\mathbf{c}' = \mathbf{b}' + \Delta \mathbf{a}',$$

where $\mathbb{F}$ is a large field, e.g. $GF(2^{128})$, and $n \approx 2k$. One party learns the values $(\mathbf{b}', \Delta)$, while the other learns $(\mathbf{a}', \mathbf{c}')$. These vectors are uniformly distributed, except that $\mathbf{a}'$ is extremely sparse. This sparsity allows for the setup to be performed with communication sublinear in the size of the vectors. In a subsequent phase, the protocol transforms $\mathbf{a}', \mathbf{b}', \mathbf{c}'$ into uniformly random looking vectors $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{F}^k$, preserving the relation

$$\mathbf{c} = \mathbf{b} + \Delta \mathbf{a}.$$

In particular, the sparse vector $\mathbf{a}' \in \mathbb{F}^n$ is transformed into a pseudorandom vector $\mathbf{a} \in \mathbb{F}^k$. For our parameter regime, k is large (e.g., 2^{20}), while $n = 2k$, and therefore this transformation is compressing.

This correlation, called *vector oblivious linear evaluation* (VOLE), underpins many state-of-the-art protocols for zero-knowledge proofs [WYKW21,YWL+20] and private set intersection [RS21,RR22]. Alternatively, one can locally convert this VOLE correlation into k instances of OT or binary Beaver triples [BCG+19a] [BCG+19b], which are employed in a wide range of cryptographic schemes.

Multiplication by $\mathbf{G}$. The core transformation step multiplies $\mathbf{a}', \mathbf{b}'$, and $\mathbf{c}'$ by a matrix $\mathbf{G} \in \mathbb{F}^{k \times n}$, where $\mathbf{G}$ is the generator matrix of an error-correcting code with high minimum distance. Namely,

$$\mathbf{a} = \mathbf{G} \cdot \mathbf{a}', \quad \mathbf{b} = \mathbf{G} \cdot \mathbf{b}', \quad \mathbf{c} = \mathbf{G} \cdot \mathbf{c}'.$$

A code $\mathbf{G}$ with high minimum distance[7] ensures that an extremely sparse $\mathbf{a}'$ is mapped to a slightly shorter but essentially uniform vector $\mathbf{a}$. More formally, under the SD (a.k.a. dual LPN) assumption associated with $\mathbf{G}$, the vector $\mathbf{a}$ is indistinguishable from a random vector given $\mathbf{G}$.

[7] There are some exceptions, such as the Reed-Solomon Code.

Improving Computational Efficiency. Two major avenues have been explored to optimize these protocols: (1) reducing the cost of multiplying by $\mathbf{G}$, which is a local computation but often the main bottleneck; and (2) minimizing the interactive setup that yields the initial correlation $\mathbf{a}' = \mathbf{b}' + \Delta\mathbf{c}'$. Our focus is on the first challenge: designing $\mathbf{G}$ so that multiplication is efficient. Although several codes have been proposed [BCG+19a,CRR21,BCG+22,RRT23], we show there is ample room for concrete improvement by constructing new high-minimum-distance codes that enable fast matrix-vector products.

A.2 Zero-Knowledge Proofs: The Blaze System

The Blaze system [BCF+25] is a state-of-the-art multilinear polynomial commitment scheme (MLPCS) designed for high-efficiency provers over binary fields. A core component of Blaze's performance is its use of linear-time encodable codes, specifically the Repeat-Accumulate-Accumulate (RAA) code [DJM98,KU98], as the underlying error-correcting primitive.

Blaze Framework. Blaze operates as a compiler that combines two primary techniques:

- **Code Interleaving:** To commit to a large multilinear polynomial of size N, the prover interprets the coefficients as a $t \times k$ matrix (where $N = tk$) and encodes each row separately using a base linear code.
- **IOPP for Multilinear Evaluation:** Blaze lifts evaluation claims about the multilinear extension into claims about simple combinatorial extensions of the base code, utilizing an inner proof system (IOPP) to verify these smaller claims.

In its standard instantiation, Blaze utilizes RAA codes at rate 1/4 to achieve the relative distance (approximately 0.19) required for soundness. While puncturing could theoretically achieve rate 1/2, it adds complexity and does not reduce the initial encoding work or memory footprint required to generate the full codeword before puncturing.

Conceptual Advantages of Block-Accumulate Codes. Our Block-Accumulate (BA-3) codes offer several conceptual and practical improvements when integrated into the Blaze framework:

- **Native Rate 1/2 Performance:** Unlike RAA codes, which require rate 1/4 for sufficient distance , BA-3 codes achieve high minimum distance (approximately 0.1) natively at rate 1/2.
- **Reduced Permutation Overhead:** By operating at rate 1/2, the permutations π_i within the code act on vectors that are half the size of those in a rate 1/4 RAA scheme. This directly reduces computational work for both the encoder and the prover.

– **Memory Efficiency and Scalability:** Blaze's claim to fame is its ability to handle massive instances ($> 2^{25}$ variables). Because BA-3 produces codewords half the size of unpunctured RAA, the total memory overhead is halved, effectively doubling the maximum polynomial size Blaze can handle on the same hardware.

Future-Proofing Prover Bottlenecks. Integrating highly optimized inner proof systems like PipFRI into Blaze highlights the critical need for faster encoding.

– **The Role of PipFRI:** PipFRI utilizes a shred-to-shine methodology that decomposes polynomials into smaller chunks, drastically reducing the overhead of Fast Fourier Transforms (FFTs) and Merkle tree construction during the commitment and opening phases. This effectively slashes the $O(N \log N)$ bottleneck of standard FRI to near-linear time.
– **The Shift in Costs:** The total prover time in Blaze is a sum of "cryptographic work" (Merkle hashing, folding, handled by the inner system) and "coding work" (encoding via RAA/BA). As PipFRI reduces the cryptographic weight by orders of magnitude (e.g., 10x faster prover times), the static cost of the "coding work" becomes the primary target for optimization.
– **The BA-3 Advantage:** By providing a 2x speedup in encoding, BA-3 lowers the static floor of the computation. This ensures that as the variable ceiling of cryptographic costs lowers with innovations like PipFRI, the encoding step does not become the new bottleneck stalling the entire pipeline.

B Block Accumulate Minimum Distance Proof

Proof overview. Fix a nonzero input $\mathbf{x} \in \{0,1\}^k$ and track Hamming weights through $\mathbf{G} = \mathbf{B}\pi_1\mathbf{A}\pi_2\mathbf{A}$:

$$u := |\mathbf{xB}|_{\mathsf{H}}, \qquad h_1 := |\mathbf{xB}\pi_1\mathbf{A}|_{\mathsf{H}}, \qquad h := |\mathbf{xB}\pi_1\mathbf{A}\pi_2\mathbf{A}|_{\mathsf{H}}.$$

We upper bound the expected number of nonzero codewords with final weight $h \le \varepsilon n$ and then apply Markov's inequality.

1. *Outer block-diagonal distance.* By (1), any weight-w input activates at least $\lceil w/\sigma \rceil$ blocks, and each active block contributes at least d' output weight. Hence $u = |\mathbf{xB}|_{\mathsf{H}} \ge u_{\min}(w) = d'\lceil w/\sigma \rceil$.
2. *Random interleavers give uniformity.* Conditioned on $|\mathbf{xB}|_{\mathsf{H}} = u$, the vector after π_1 is uniform among all length-n vectors of weight u; similarly, conditioned on h_1, the vector after π_2 is uniform among all weight-h_1 vectors. Thus both accumulator stages are analyzed via the exact input-output enumerator $\mathsf{E}_{u,h}$ of the accumulator.
3. *Accumulators suppress very small outputs.* Lemma 2 shows that for $h_1 \le 2\varepsilon n$,

$$\Pr[|z\mathbf{A}|_{\mathsf{H}} = h_1 \mid |z|_{\mathsf{H}} = u] \;\le\; \left(4e^2 \frac{h_1}{n}\right)^{u/2},$$

and the RHS is monotone decreasing in u when ε is small. This lets us eliminate the (potentially complicated) distribution of u given $\mathbf{x}$ and upper bound by the worst case allowed by (1). Applying the same estimate to the second accumulator yields a strong tail bound on $\Pr[h \le \varepsilon n \mid h_1]$.

4. *Support constraints truncate the weight ranges.* Since $\mathsf{E}_{a,b} = 0$ unless $\lceil a/2 \rceil \le b$, any final weight $h \le \varepsilon n$ forces $h_1 \le 2\varepsilon n$ and in turn forces $u \le 2h_1$. Combined with $u \ge u_{\min}(w)$, this restricts the relevant inputs to a low-weight regime.
5. *Re-index by the number of active blocks.* The key counting step is to sum over inputs by the number b of active σ-bit blocks rather than by total Hamming weight w. For inputs with exactly b active blocks we have the sharp bound $u \ge d'b$, and the number of such inputs is at most $\binom{k/\sigma}{b}(2^\sigma - 1)^b$. This block-based counting aligns with the structure of $\mathbf{B}$ and yields a geometric decay in b once $d' > 4$, implying that the expected number of codewords with $h \le \varepsilon n$ tends to zero.

Assumption 1 (Outer block-diagonal distance)**.** *Assume* $\mathbf{B}$ *is block-diagonal with input-block size* σ *and that each activated block produces output Hamming weight at least some constant* $d' > 4$*. Then for any* $\mathbf{x}$ *with* $w := |\mathbf{x}|_{\mathsf{H}}$*,*

$$u := |\mathbf{x}\mathbf{B}|_{\mathsf{H}} \;\ge\; u_{\min}(w) := d' \left\lceil \frac{w}{\sigma} \right\rceil. \tag{1}$$

Accumulator input-output enumerator. Using the standard accumulator from the preliminaries, we recall the (exact) input–output enumerator:

$$\mathsf{E}^{\mathbf{A}}_{u,h} = \binom{n-h}{\lfloor u/2 \rfloor} \binom{h-1}{\lceil u/2 \rceil - 1}. \tag{2}$$

We will omit the superscript $\mathbf{A}$ in this section. In particular, for uniform $z \in \{0,1\}^n$ conditioned on $|z|_{\mathsf{H}} = u$,

$$\Pr[|z\mathbf{A}|_{\mathsf{H}} = h \mid |z|_{\mathsf{H}} = u] = \frac{\mathsf{E}_{u,h}}{\binom{n}{u}}, \tag{3}$$

and $\mathsf{E}_{u,h} = 0$ unless $\lceil u/2 \rceil \le h$.

Lemma 2 (Small-output contraction for the accumulator). *Let* $\mathbf{A}$ *be the length-*n *accumulator with enumerator* (2)*. There exists a constant* $\varepsilon_0 > 0$ *such that for all* $0 < \varepsilon \le \varepsilon_0$*, all sufficiently large* n*, all* $1 \le h_1 \le 2\varepsilon n$*, and all* $u \ge 1$*,*

$$\Pr\left[|z\mathbf{A}|_{\mathsf{H}} = h_1 \mid |z|_{\mathsf{H}} = u\right] = \frac{\mathsf{E}_{u,h_1}}{\binom{n}{u}} \le \left(4e^2 \frac{h_1}{n}\right)^{u/2}. \tag{4}$$

Moreover, if $8e^2\varepsilon < 1$*, then the RHS of* (4) *is decreasing in* u*.*

In addition, for each fixed $h_1 \le 2\varepsilon n$*,*

$$\sum_{h=1}^{\lfloor \varepsilon n \rfloor} \Pr\left[|z\mathbf{A}|_{\mathsf{H}} = h \mid |z|_{\mathsf{H}} = h_1\right] \le (\varepsilon n)\left(4e^2\varepsilon\right)^{h_1/2}. \tag{5}$$

Proof. We prove (4) first. Let $s = \lfloor u/2 \rfloor$ and $t = \lceil u/2 \rceil$, so $s + t = u$ and $s, t \geq u/2 - 1/2$. Using (2), $\mathsf{E}_{u,h_1} = \binom{n-h_1}{s}\binom{h_1-1}{t-1} \leq \binom{n}{s}\binom{h_1}{t}$. Apply $\binom{N}{r} \leq (eN/r)^r$ to both factors: $\mathsf{E}_{u,h_1} \leq \left(\frac{en}{s}\right)^s \left(\frac{eh_1}{t}\right)^t$. Also use $\binom{n}{u} \geq (n/u)^u$. Then $\frac{\mathsf{E}_{u,h_1}}{\binom{n}{u}} \leq \left(\frac{en}{s}\right)^s \left(\frac{eh_1}{t}\right)^t \left(\frac{u}{n}\right)^u = \left(e\frac{u}{s}\right)^s \left(e\frac{u}{t}\right)^t \left(\frac{h_1}{n}\right)^t$. If $u = 1$, then $\mathsf{E}_{1,h_1} = 1$ and $\binom{n}{1} = n$, so $\Pr[|z\mathbf{A}|_{\mathsf{H}} = h_1 \mid |z|_{\mathsf{H}} = 1] = 1/n \leq (4e^2(h_1/n))^{1/2}$ for all sufficiently large n. Assume henceforth that $u \geq 2$, so $s, t \geq 1$. If u is even then $s = t = u/2$ and $(u/s)^s(u/t)^t = 2^u$. If u is odd, write $u = 2m + 1$ so $s = m$ and $t = m + 1$, and observe that

$$\left(\frac{u}{s}\right)^s \left(\frac{u}{t}\right)^t = 2^u \left(1 + \frac{1}{2m}\right)^m \left(1 - \frac{1}{2m+2}\right)^{m+1} \leq 2^u,$$

where the last inequality follows from $\log(1+x) \leq x$ and $\log(1-x) \leq -x - x^2/2$. Thus $\left(e\frac{u}{s}\right)^s \left(e\frac{u}{t}\right)^t \leq (2e)^u$, and hence

$$\frac{\mathsf{E}_{u,h_1}}{\binom{n}{u}} \leq (2e)^u \left(\frac{h_1}{n}\right)^t \leq (2e)^u \left(\frac{h_1}{n}\right)^{u/2}.$$

Finally, $(2e)^u = (4e^2)^{u/2}$ gives (4). If $h_1 \leq 2\varepsilon n$ and $8e^2\varepsilon < 1$, then $4e^2(h_1/n) \leq 8e^2\varepsilon < 1$, so the RHS of (4) is decreasing in u.

For (5), apply (4) with $u \leftarrow h_1$ to get, for each h, $\Pr[|z\mathbf{A}|_{\mathsf{H}} = h \mid |z|_{\mathsf{H}} = h_1] \leq \left(4e^2\frac{h}{n}\right)^{h_1/2}$. Summing over $1 \leq h \leq \varepsilon n$ and using $h/n \leq \varepsilon$ yields

$$\sum_{h=1}^{\lfloor \varepsilon n \rfloor} \Pr[|z\mathbf{A}|_{\mathsf{H}} = h \mid |z|_{\mathsf{H}} = h_1] \leq (\varepsilon n)\left(4e^2\varepsilon\right)^{h_1/2}.$$

□

Theorem 2 (Linear minimum distance for BA-3). *Consider the Block-Accumulate code $\mathbf{G} := \mathbf{B}\pi_1\mathbf{A}\pi_2\mathbf{A}$ of length n, where π_1, π_2 are independent uniform interleavers on $[n]$ and $\mathbf{A}$ is the standard length-n accumulator. Assume the outer block-diagonal distance lower bound (1) holds. If the outer block distance satisfies $d' > 4$, then for some sufficiently small constant $\varepsilon = \varepsilon(d', \sigma, R) > 0$,*

$$\Pr[d_{\min}(\mathbf{G}) \geq \varepsilon n] \xrightarrow[n\to\infty]{} 1.$$

Remark 1. We write $R := k/n$ for the (constant) rate of the code. Here ε depends only on d', σ and the rate R, and not on n.

Proof. **Step 1: Expected number of low-weight codewords.**

Fix a constant $0 < \varepsilon \leq \varepsilon_0$ sufficiently small so that $8e^2\varepsilon < 1$, $(4e^2\varepsilon)^{1/2}e < 1$, and $\rho < 1$ (as defined later). Let $E_{\leq \varepsilon n}$ denote the expected number of *nonzero* codewords of weight at most εn. By linearity of expectation,

$$E_{\leq \varepsilon n} = \sum_{w=1}^{k} \binom{k}{w} \Pr\left[|\mathbf{x}\mathbf{G}|_{\mathsf{H}} \leq \varepsilon n\right], \tag{6}$$

where $\mathbf{x}$ is uniform over $\{x \in \{0,1\}^k : |x|_{\mathsf{H}} = w\}$.

Introduce the intermediate weights

$$u = |\mathbf{xB}|_{\mathsf{H}}, \qquad h_1 = |\mathbf{xB}\pi_1\mathbf{A}|_{\mathsf{H}}, \qquad h = |\mathbf{xB}\pi_1\mathbf{A}\pi_2\mathbf{A}|_{\mathsf{H}} = |\mathbf{xG}|_{\mathsf{H}} = |\mathbf{y}|_{\mathsf{H}}.$$

Conditioning on (u, h_1, h),

$$E_{\le \varepsilon n} = \sum_{w=1}^{k} \binom{k}{w} \sum_{u=0}^{n} \sum_{h_1=0}^{n} \sum_{h=1}^{\lfloor \varepsilon n \rfloor} \Pr[u \mid w]\ \Pr[h_1 \mid u]\ \Pr[h \mid h_1]. \tag{7}$$

Uniformity under random interleavers. Conditioned on $|\mathbf{xB}|_{\mathsf{H}} = u$, the vector $\mathbf{xB}\pi_1$ is uniform over all weight-u vectors in $\{0,1\}^n$; similarly, conditioned on h_1, the vector $\mathbf{xB}\pi_1\mathbf{A}\pi_2$ is uniform over all weight-h_1 vectors in $\{0,1\}^n$. Thus (3) applies to both accumulator stages:

$$\Pr[h_1 \mid u] = \frac{\mathsf{E}_{u,h_1}}{\binom{n}{u}}, \qquad \Pr[h \mid h_1] = \frac{\mathsf{E}_{h_1,h}}{\binom{n}{h_1}}. \tag{8}$$

Step 2: Remove the distribution of u using monotonicity.

We only know $u \ge u_{\min}(w)$ from (1); the exact distribution $\Pr[u \mid w]$ may be complicated. We eliminate it using a one-sided bound.

For $h_1 \le 2\varepsilon n$, Lemma 2(i) gives $\Pr[|z\mathbf{A}|_{\mathsf{H}} = h_1 \mid |z|_{\mathsf{H}} = u] = \frac{\mathsf{E}_{u,h_1}}{\binom{n}{u}} \le \left(4e^2 \frac{h_1}{n}\right)^{u/2}$. If $8e^2\varepsilon < 1$, then $4e^2(h_1/n) \le 8e^2\varepsilon < 1$ and the RHS is decreasing in u. Hence, for all $u \ge u_{\min}(w)$, $\Pr[|z\mathbf{A}|_{\mathsf{H}} = h_1 \mid |z|_{\mathsf{H}} = u] \le \left(4e^2 \frac{h_1}{n}\right)^{u_{\min}(w)/2}$. Therefore, using $\sum_u \Pr[u \mid w] = 1$ and the fact that $u \ge u_{\min}(w)$, we may upper bound the u-average by the worst case: $\sum_u \Pr[u \mid w] \Pr[h_1 \mid u] \le \left(4e^2 \frac{h_1}{n}\right)^{u_{\min}(w)/2}$ for $(h_1 \le 2\varepsilon n)$.

Substituting into (7) and using (8) for the second accumulator yields

$$E_{\le \varepsilon n} \le \sum_{w=1}^{k} \binom{k}{w} \sum_{h_1=0}^{n} \sum_{h=1}^{\lfloor \varepsilon n \rfloor} \left(4e^2 \frac{h_1}{n}\right)^{u_{\min}(w)/2} \cdot \frac{\mathsf{E}_{h_1,h}}{\binom{n}{h_1}}. \tag{9}$$

Step 3: Truncation (support constraints).

From (2), $\mathsf{E}_{a,b} = 0$ unless $\lceil a/2 \rceil \le b$. Thus $\frac{\mathsf{E}_{h_1,h}}{\binom{n}{h_1}} = 0$ unless $h_1 \le 2h$. Since $h \le \varepsilon n$, we may restrict to

$$1 \le h_1 \le 2\varepsilon n. \tag{10}$$

Also, the first stage has $\mathsf{E}_{u,h_1} = 0$ unless $u \le 2h_1$; since $u \ge u_{\min}(w)$, we must have $u_{\min}(w) \le 2h_1$, i.e. $d' \left\lceil \frac{w}{\sigma} \right\rceil \le 2h_1 \quad \Rightarrow \quad w \le \sigma \left\lfloor \frac{2h_1}{d'} \right\rfloor$. So (9) becomes

$$E_{\le \varepsilon n} \le \sum_{h_1=1}^{\lfloor 2\varepsilon n \rfloor} \sum_{w=1}^{\sigma \lfloor 2h_1/d' \rfloor} \binom{k}{w} \left(4e^2 \frac{h_1}{n}\right)^{u_{\min}(w)/2} \sum_{h=1}^{\lfloor \varepsilon n \rfloor} \frac{\mathsf{E}_{h_1,h}}{\binom{n}{h_1}}. \tag{11}$$

Step 4: Bound the second-stage tail $\sum_{h \le \varepsilon n} \Pr[h \mid h_1]$.

By Lemma 2(i) with $u \leftarrow h_1$,

$$\frac{\mathsf{E}_{h_1,h}}{\binom{n}{h_1}} \le \left(4e^2\frac{h}{n}\right)^{h_1/2}.$$

Therefore,

$$\sum_{h=1}^{\lfloor \varepsilon n \rfloor} \frac{\mathsf{E}_{h_1,h}}{\binom{n}{h_1}} \le \sum_{h=1}^{\lfloor \varepsilon n \rfloor} \left(4e^2\frac{h}{n}\right)^{h_1/2} \le (\varepsilon n)\left(4e^2\varepsilon\right)^{h_1/2}, \tag{12}$$

where the last step uses monotonicity in h and $h \le \varepsilon n$.

Substitute (12) into (11):

$$E_{\le \varepsilon n} \le (\varepsilon n) \sum_{h_1=1}^{\lfloor 2\varepsilon n \rfloor} \left(4e^2\varepsilon\right)^{h_1/2} \sum_{w=1}^{\sigma\lfloor 2h_1/d' \rfloor} \binom{k}{w} \left(4e^2\frac{h_1}{n}\right)^{u_{\min}(w)/2}. \tag{13}$$

Step 5: Re-indexing by active blocks.

For an input $\mathbf{x} \in \{0,1\}^k$, let $b = b(\mathbf{x}) \in \{1, \dots, k/\sigma\}$ denote the number of σ-bit input blocks on which $\mathbf{x}$ is nonzero. Since each active block contributes output weight at least d', we have

$$u = |\mathbf{x}\mathbf{B}|_{\mathsf{H}} \ \ge \ d'b. \tag{14}$$

For fixed b, the number of inputs activating exactly b blocks is at most

$$N_b \ \le \ \binom{k/\sigma}{b}(2^\sigma - 1)^b, \tag{15}$$

since we choose the b active blocks and then choose any nonzero σ-bit pattern in each active block.

We now bound $E_{\le \varepsilon n}$ by summing over inputs grouped by the number $b = b(\mathbf{x})$ of active σ-bit blocks (rather than by total Hamming weight). From (6), equivalently

$$E_{\le \varepsilon n} = \sum_{\mathbf{x} \in \{0,1\}^k \setminus \{0\}} \Pr\big[|\mathbf{x}\mathbf{G}|_{\mathsf{H}} \le \varepsilon n\big].$$

Fix such an input $\mathbf{x}$ and let $b = b(\mathbf{x})$. By (14) we have $u = |\mathbf{x}\mathbf{B}|_{\mathsf{H}} \ge d'b$. Using the support constraint $\mathsf{E}_{u,h_1} = 0$ unless $u \le 2h_1$ and applying (12), we obtain

$$\Pr\big[|\mathbf{x}\mathbf{G}|_{\mathsf{H}} \le \varepsilon n\big] \le (\varepsilon n) \sum_{h_1=1}^{\lfloor 2\varepsilon n \rfloor} \left(4e^2\varepsilon\right)^{h_1/2} \Pr[h_1 \mid u].$$

For each $h_1 \le 2\varepsilon n$, Lemma 2(i) gives

$$\Pr[h_1 \mid u] = \frac{\mathsf{E}_{u,h_1}}{\binom{n}{u}} \le \left(4e^2\frac{h_1}{n}\right)^{u/2} \le \left(4e^2\frac{h_1}{n}\right)^{d'b/2},$$

where the last inequality uses $u \geq d'b$ and $4e^2(h_1/n) \leq 8e^2\varepsilon < 1$. If $d'b > 2h_1$ then $\Pr[h_1 \mid u] = 0$, so only $b \leq 2h_1/d'$ contribute. Summing over all inputs with exactly b active blocks and using (15) gives

$$E_{\leq \varepsilon n} \leq (\varepsilon n) \sum_{h_1=1}^{\lfloor 2\varepsilon n \rfloor} \left(4e^2\varepsilon\right)^{h_1/2} \sum_{b=1}^{\lfloor 2h_1/d' \rfloor} N_b \left(4e^2 \frac{h_1}{n}\right)^{d'b/2}. \tag{16}$$

(5b) Bounding the h_1-sum and geometric decay in b.

We now swap the order of summation. Since $b \leq 2h_1/d'$ if and only if $h_1 \geq d'b/2$, (16) becomes

$$E_{\leq \varepsilon n} \leq (\varepsilon n) \sum_{b=1}^{\lfloor 4\varepsilon n/d' \rfloor} N_b \sum_{h_1=\lceil d'b/2 \rceil}^{\lfloor 2\varepsilon n \rfloor} \left(4e^2\varepsilon\right)^{h_1/2} \left(4e^2 \frac{h_1}{n}\right)^{d'b/2}. \tag{17}$$

Define $f(h_1) := \left(4e^2\varepsilon\right)^{h_1/2} \left(4e^2 \frac{h_1}{n}\right)^{d'b/2}$. For $h_1 \geq d'b/2$, we bound the ratio $\frac{f(h_1+1)}{f(h_1)} = (4e^2\varepsilon)^{1/2} \left(1 + \frac{1}{h_1}\right)^{d'b/2} \leq (4e^2\varepsilon)^{1/2} \exp\left(\frac{d'b}{2h_1}\right) \leq (4e^2\varepsilon)^{1/2} e$, where the last inequality uses $h_1 \geq d'b/2$. Choosing $\varepsilon > 0$ sufficiently small so that $(4e^2\varepsilon)^{1/2} e < 1$, the inner sum is geometrically decreasing, and there exists a constant C_0 (independent of n and b) such that

$$\sum_{h_1=\lceil d'b/2 \rceil}^{\lfloor 2\varepsilon n \rfloor} f(h_1) \;\leq\; C_0\, f(\lceil d'b/2 \rceil). \tag{18}$$

Using $\lceil d'b/2 \rceil \leq d'b$ and $h_1/n \leq d'b/n$ at the lower limit, we obtain $f(\lceil d'b/2 \rceil) \leq \left(4e^2\varepsilon\right)^{d'b/4} \left(4e^2 \frac{d'b}{n}\right)^{d'b/2}$. Substituting into (17) yields

$$E_{\leq \varepsilon n} \leq (\varepsilon n)\, C_0 \sum_{b=1}^{\lfloor 4\varepsilon n/d' \rfloor} N_b \left(4e^2\varepsilon\right)^{d'b/4} \left(4e^2 \frac{d'b}{n}\right)^{d'b/2}. \tag{19}$$

Finally, using (15) and $\binom{k/\sigma}{b} \leq (ek/(\sigma b))^b$ with $k = Rn$, we obtain

$$E_{\leq \varepsilon n} \leq (\varepsilon n)\, C_0 \sum_{b=1}^{\lfloor 4\varepsilon n/d' \rfloor} \left[\frac{eR(2^\sigma - 1)}{\sigma b} \cdot \left(4e^2\varepsilon\right)^{d'/4} \cdot \left(4e^2 d'b\right)^{d'/2} \cdot n^{\,1-d'/2}\right]^b.$$

We now analyze the b-sum. Write the final bound as

$$E_{\leq \varepsilon n} \leq (\varepsilon n)\, C_0 \sum_{b=1}^{\lfloor 4\varepsilon n/d' \rfloor} \left[K(\sigma, R, d', \varepsilon) \cdot b^{\,d'/2-1} \cdot n^{\,1-d'/2}\right]^b,$$

where

$$K(\sigma, R, d', \varepsilon) := \frac{eR(2^\sigma - 1)}{\sigma} \cdot \left(4e^2\varepsilon\right)^{d'/4} \cdot \left(4e^2 d'\right)^{d'/2}$$

absorbs all factors independent of b and n.

The bottleneck is $b = 1$. The $b = 1$ term contributes

$$E_{\le \varepsilon n}^{(b=1)} \;\le\; (\varepsilon n)\, C_0 \cdot K(\sigma, R, d', \varepsilon) \cdot n^{\,1-d'/2} = O\Big(n^{\,2-d'/2}\Big)\,.$$

Therefore, to ensure $E_{\le \varepsilon n} \to 0$ as $n \to \infty$, we require

$$2 - \frac{d'}{2} < 0 \qquad \Longleftrightarrow \qquad d' > 4.$$

The terms with $b \ge 2$ are negligible. Let $\alpha := d'/2 - 1 > 1$. The bth summand equals

$$\left\lceil K(\sigma, R, d', \varepsilon) \left(\frac{b}{n}\right)^{\alpha} \right\rceil^{b}.$$

For all sufficiently large n (so that $\sqrt{n} \le 4\varepsilon n/d'$), split the range into $2 \le b \le \lfloor \sqrt{n} \rfloor$ and $b \ge \lceil \sqrt{n} \rceil$.

If $2 \le b \le \sqrt{n}$ then $b/n \le n^{-1/2}$, so

$$\left\lceil K \left(\frac{b}{n}\right)^{\alpha} \right\rceil^{b} \le \Big(K\, n^{-\alpha/2}\Big)^{b},$$

and hence, for all sufficiently large n (so that $K\, n^{-\alpha/2} < 1/2$),

$$\sum_{b=2}^{\lfloor \sqrt{n} \rfloor} \left\lceil K \left(\frac{b}{n}\right)^{\alpha} \right\rceil^{b} \le \sum_{b=2}^{\infty} \Big(K\, n^{-\alpha/2}\Big)^{b} = O\big(n^{-\alpha}\big)\,.$$

For $b \ge \sqrt{n}$, use the truncation $b \le \lfloor 4\varepsilon n/d' \rfloor$ to get $b/n \le 4\varepsilon/d'$. Define

$$\rho := K(\sigma, R, d', \varepsilon) \left(\frac{4\varepsilon}{d'}\right)^{\alpha},$$

and choose ε sufficiently small so that $\rho < 1$. Then for all $b \ge \sqrt{n}$,

$$\left\lceil K \left(\frac{b}{n}\right)^{\alpha} \right\rceil^{b} \le \rho^{b} \le \rho^{\sqrt{n}},$$

so

$$\sum_{b=\lceil \sqrt{n} \rceil}^{\lfloor 4\varepsilon n/d' \rfloor} \left\lceil K \left(\frac{b}{n}\right)^{\alpha} \right\rceil^{b} \le \frac{\rho^{\sqrt{n}}}{1-\rho} = o(n^{-2}).$$

Multiplying by the outer factor $(\varepsilon n) C_0$ shows that the total contribution of $b \ge 2$ is $o(1)$. Together with the $b = 1$ estimate above, this implies $E_{\le \varepsilon n} \to 0$ as $n \to \infty$.

Finally, by Markov's inequality,

$$\Pr[d_{\min}(\mathbf{G}) \le \varepsilon n] \le E_{\le \varepsilon n} \xrightarrow[n \to \infty]{} 0,$$

which proves the theorem.

□

C Balls in Bins with Slots

The k labeled slotted bins, each with a capacity of c, collectively provide a total of ck slots. There are $\binom{ck}{n}$ ways to assign n identical balls into ck slots. However, this includes assignments where bins may be empty. Let A_Y denote the set of assignments where the bins in $Y \subseteq [k]$ are empty, i.e. $\mathbf{x}_y = 0^c$ for $y \in Y$. Note that by definition,

$$\mathsf{S}(n,k,c) = \binom{ck}{n} - \left|\bigcup_{i\in[k]} A_{\{i\}}\right|.$$

From the inclusion-exclusion principle,

$$\left|\bigcup_{i\in[k]} A_{\{i\}}\right| = \sum_{i\in[k]} |A_{\{i\}}| - \sum_{\substack{i,i'\in[k]\\ i\neq i'}} |A_{\{i,i''\}}| + \sum_{\substack{i,i',i''\in[k]\\ i\neq i',i\neq i'',i'\neq i''}} |A_{\{i,i',i''\}}| - \ldots,$$

or equivalently,

$$\left|\bigcup_{i\in[k]} A_{\{i\}}\right| = \sum_{i=1}^{k} (-1)^{i-1} \sum_{\substack{Y\subseteq[k]\\ |Y|=i}} |A_Y|.$$

For any $Y \subseteq [k]$,

$$|A_Y| = \binom{c(k-|Y|)}{n},$$

as if the bins in Y must be empty, the n balls must be assigned to the remaining $c(k-|Y|)$ slots in the $k-|Y|$ bins. Furthermore, the number of $Y \subseteq [k]$ with $|Y| = i$ is $\binom{k}{i}$. Therefore,

$$\left|\bigcup_{i\in[k]} A_{\{i\}}\right| = \sum_{i=1}^{k} (-1)^{i-1} \binom{k}{i} \binom{c(k-i)}{n},$$

and hence

$$\mathsf{S}(n,k,c) = \binom{ck}{n} - \sum_{i=1}^{k} (-1)^{i-1} \binom{k}{i} \binom{c(k-i)}{n} = \sum_{i=0}^{k} (-1)^{i} \binom{k}{i} \binom{c(k-i)}{n},$$

as required.

D Repeat-Accumulate Codes

Repeat-Accumulate (RA-3) codes, where $\mathbf{B}$ is replaced by a repeater, achieve a concretely high linear minimum distance at rate 1/4. For efficiency reasons, we focus on rate 1/2 to avoid the high cost of large permutations. We implement the code for rate 1/2 and depict the result in Figure 11 for $k \in \{512, 1024, 2048\}$. As

can be seen, the probability of having a weight-$h = 1$ codeword is approximately $1/k$ and increases with h until around $h/n = 0.03$, at which point we expect there to be a single codeword. While the asymptotic properties of rate-1/2 RA-3 codes remain unclear, our experiments show their concrete minimum distance is poor.

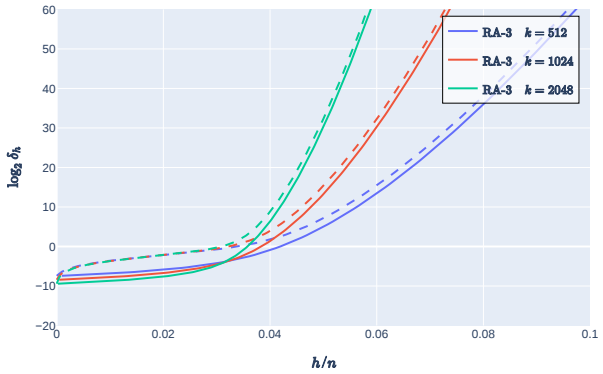


Fig. 11. Depictions of the Repeat-Accumulate enumerator. The dashed lines denote the log number of codewords with at most h/n relative minimum distance, as opposed to exactly h/n. See Figure 1 for details.

E Fully Systematic Block-Diagonal

A common technique in coding theory is to describe a systematic code where the generator $\mathbf{G}$ can be expressed as $\mathbf{G} := (\mathbf{I}||\mathbf{G}')$. This approach can be more efficient because the codeword contains the input as its first part. Therefore, one needs only to compute the remaining part of the codeword. While any arbitrary code can be converted into systematic form, such a conversion is typically costly, as it requires Gaussian elimination and eliminates the possibility of fast encoding.

[RRT23] and others have shown that it can be possible to instantiate codes already in systematic form such that the computation of $\mathbf{xG}'$ retains the fast encoding structure. In particular, the idea is to simply instantiate $\mathbf{G}'$ as an instance of the structured code, in our case a serially concatenated Block-Diagonal code. If we restrict our attention to rate 1/2 codes, this implies that $\mathbf{G}'$ is a square $k \times k$ matrix that can be expressed as $\mathbf{G}' := \mathbf{G}'_1\pi_1\mathbf{G}'_2...\pi_{t-1}\mathbf{G}'_t$, where each $\mathbf{G}'_i$ is a Block-Diagonal matrix. We observe that one must approximately double the size of σ to maintain the same minimum distance behavior. This doubling results in four times the amount of work to multiply but halves the amount of work to permute. Based on the current running time numbers we conclude this approach does not result in an overall reduction in running time.

E.1 Block-Diagonal Weight Distribution at $t \geq 2$

More generally, we observe that the phase transition occurs approximately at

$$\log(\sigma) = \frac{\log(k)}{t} + 3/4, \quad \sigma = 2^{3/4}\sqrt[t]{k}$$

This implies that at $t = \log(k)$ a submatrix of size $\sigma = O(1)$ is sufficient. The resulting weight distributions are largely similar to the case of $t = 2$ discussed in Section 8.1. We plot the measured values of where the phase transition intersects $\log \delta_h = 0$ in Figure 12 for $t \in [2, 5]$ along with the trend lines $\sigma = 2^{3/4}\sqrt[t]{k}$. While there is some variance between the trend lines and the measured phase transition, the overall fit is extremely tight. We conjecture that this general trend continues for larger values of t but leave the asymptotic analysis of the growth to future work. From a practical perspective, values beyond $t = 4$ do not offer compelling improvements given that the cost of t permutations dominates the cost of a block diagonal matrix multiply for such small values of σ.

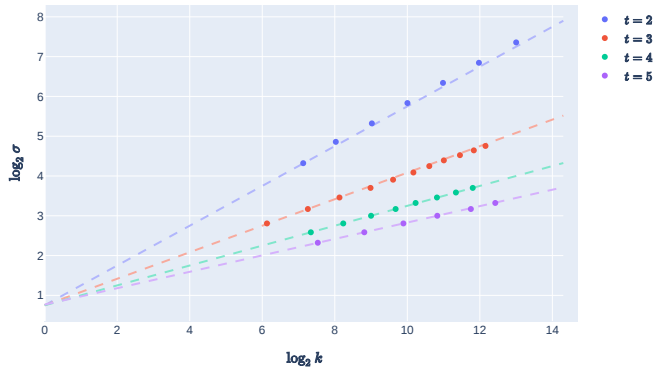


Fig. 12. Plot of the values of σ required for the phase transition points to occur at $\log_2 \delta_h = 0$, compared the the trend lines $\sigma = 2^{3/4}\sqrt[t]{k}$, equivalently $\log_2 \sigma = \frac{\log_2 k}{t} + 3/4$.

If we fix $k \approx 2^{10}$ and set $\sigma = 2^{3/4}\sqrt[t]{k}$ we obtain the plot in Figure 13. As can be see, the phase transition does occur at $\log_2 \delta_h = 0$. Interestingly, as t increases and σ decrease, we observe the the phase transition happens at smaller values of h/n. It is not immediately clear why this occurs but we note that increasing σ does reduce this effect.

More importantly, for larger values of t the probability of getting low weight codewords does not fall as quickly as the case of $t = 2$. This raises the possibility of having significantly lower minimum distance than compared to where the weight distribution crosses $\log_2 \delta_h = 0$. In particular, in Figure 13 the cumulative weight distribution is plotted as a dashed line where for each h/n value we plot

the expected number of codewords with *at most* h/n relative Hamming weight. This more accurately represents the minimum distance, as where this line crosses the $\log \delta_h = 0$ denotes the expected minimum distance. Therefore, we would desire a formula for σ such that the relative minimum distance remains close to $h/n = 0.1$. Figure 14 plots the cumulative weight distribution when setting $\sigma = 2\sqrt[t]{k}$ which does obtains the desired effect at $t \in \{4, 5\}$. For $t \in \{2, 3\}$ this parameterization overshoots and results in a cumulative phase transition below $\log \delta_h = 2$. Regardless, the general trend is apparent, one can set $\sigma = c\sqrt[t]{k}$ for some constant c such that the desired security margin can be obtained. We believe $t \in \{2, 3\}$ are the most interesting cases for which $c = 3/4$ and $c = 2$ suffices in most applications, respectively.

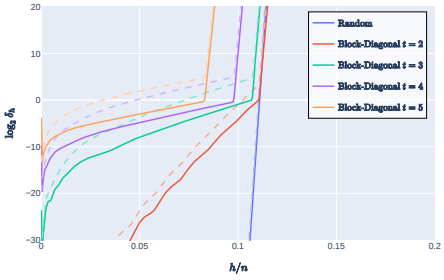


Fig. 13. Weight distribution for $t \in [2, 5]$, $k \approx 2^{10}$ and $\sigma = 2^{3/4}\sqrt[t]{k}$. Dashed lines denote the cumulative weight distribution.

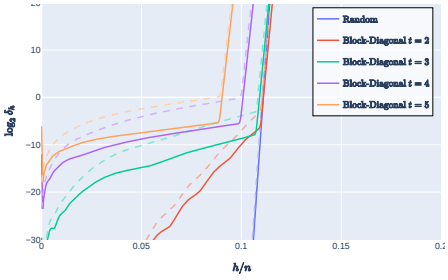


Fig. 14. Weight distribution for $t \in [2, 5]$, $k \approx 2^{10}$ and $\sigma = 2\sqrt[t]{k}$. Dashed lines denote the cumulative weight distribution.

F Linear Minimum Distance of Pure Block-Diagonal

This subsection provides a self-contained proof template showing that pure block-diagonal serial codes achieve *linear* minimum distance at rate $R = 1/2$ when the block size σ grows as a suitable power of n. The argument is entirely elementary: it combines (i) binomial concentration for the weight growth across random linear blocks, (ii) an occupancy bound for how a random permutation spreads mass across σ-blocks, and (iii) a union bound organized by the message *block-type* $a(x)$ (the number of nonzero message blocks).

We give the full proof for depth $t = 3$, which is the most delicate constant-depth case: the analysis must explicitly handle the smallest block-types, which induce an inherent $\exp(-\Theta(\sigma))$ "floor" in the failure probability. For every constant depth $t \geq 4$, the same steps go through with additional slack because the extra (π_i, G_{i+1}) layers provide further amplification before the final binomial tail; we summarize the necessary modifications after the $t = 3$ proof.

F.1 Notation and Conventions

Throughout, log denotes the natural logarithm and $\log_2$ denotes base-2. Let $H_2(p) := -p\log_2 p - (1-p)\log_2(1-p)$ be the binary entropy (base 2), and let $h(p) := -p\log p - (1-p)\log(1-p)$ be the natural entropy.

Assume $\sigma \mid n$ and $\sigma \mid k$ and $k = n/2$. Write

$$q := \frac{n}{\sigma}, \qquad b := \frac{k}{\sigma} = \frac{q}{2}.$$

Partition length-n vectors into q contiguous blocks of length σ and define $A(z)$ to be the number of nonzero blocks of z. Partition $x \in \mathbb{F}_2^k$ into b message blocks of length σ and let $a(x)$ be the number of nonzero message blocks.

The depth-3 pure block-diagonal serial ensemble is

$$G \;=\; G_1\pi_1 G_2\pi_2 G_3 \in \mathbb{F}_2^{k\times n},$$

where:

- G_1 is block-diagonal with b independent blocks, each uniform in $\mathbb{F}_2^{\sigma\times 2\sigma}$;
- G_2, G_3 are block-diagonal with q independent blocks, each uniform in $\mathbb{F}_2^{\sigma\times\sigma}$;
- π_1, π_2 are independent uniform permutations of $[n]$.

For $x \in \mathbb{F}_2^k$, define the intermediate vectors

$$v_1 := xG_1, \quad u_1 := v_1\pi_1, \quad v_2 := u_1G_2, \quad u_2 := v_2\pi_2, \quad y := u_2G_3 = xG.$$

F.2 Elementary Lemmas

We use only the following standard facts.

Lemma 3 (Uniform linear image). *Let $M \leftarrow \mathbb{F}_2^{r\times m}$ be uniform (i.i.d. entries) and fix any $u \in \mathbb{F}_2^r \setminus \{0\}$. Then uM is uniform in $\mathbb{F}_2^m$. In particular, $|uM| \sim \mathrm{Binomial}(m, 1/2)$.*

Proof. The map $\phi : \mathbb{F}_2^{r\times m} \to \mathbb{F}_2^m$ given by $\phi(M) = uM$ is a surjective $\mathbb{F}_2$-linear map because $u \neq 0$ implies there exists i with $u_i = 1$, and by choosing the ith row of M arbitrarily one can obtain any output. A uniform input to a surjective linear map yields a uniform output. Each coordinate of uM is a nontrivial $\mathbb{F}_2$-linear form in the entries of M, hence unbiased; independence follows because distinct coordinates depend on disjoint sets of column entries. Thus uM is uniform and its Hamming weight is $\mathrm{Binomial}(m, 1/2)$. □

Lemma 4 (Binomial lower tail via entropy). *Let $Z \sim \mathrm{Binomial}(M, 1/2)$. Then for every $\rho \in (0, 1/2)$,*

$$\Pr(Z \le \rho M) \le 2^{-M\,(1-H_2(\rho))}.$$

Proof. For $\rho < 1/2$,

$$\Pr(Z \le \rho M) = \sum_{i=0}^{\lfloor \rho M \rfloor} \binom{M}{i} 2^{-M} \le (\lfloor \rho M \rfloor + 1) \binom{M}{\lfloor \rho M \rfloor} 2^{-M}.$$

Using the standard bound $\binom{M}{\rho M} \le 2^{M H_2(\rho)}$ (for $\rho M \in Z$; otherwise use floor/ceiling), we obtain $\Pr(Z \le \rho M) \le \mathrm{poly}(M) \cdot 2^{-M(1-H_2(\rho))}$. Absorbing the polynomial factor into a slightly weaker constant for large M yields the stated inequality. (If desired, one can remove $\mathrm{poly}(M)$ by a slightly sharper counting argument; we do not need it.) □

Lemma 5 (Type count). *For each $a \in \{0, \dots, b\}$,*

$$\#\{x \in \mathbb{F}_2^k : \ a(x) = a\} = \binom{b}{a} (2^\sigma - 1)^a \le \left(\frac{eb}{a}\right)^a 2^{\sigma a}.$$

Proof. Choose which a message blocks are nonzero (at most $\binom{b}{a}$ choices), and for each chosen block choose any nonzero vector in $\mathbb{F}_2^\sigma$ (exactly $2^\sigma - 1$ choices). Finally apply $\binom{b}{a} \le (eb/a)^a$ and $2^\sigma - 1 \le 2^\sigma$. □

Lemma 6 (Occupancy bound for a uniform permutation). *Let $S \in \{0, 1, \dots, n\}$ and let $W \subseteq [n]$ be a uniformly random S-subset. Partition $[n]$ into q blocks of size σ. Let B be the number of blocks that intersect W (equivalently, the number of nonzero blocks in the permuted vector). Then for every integer $m \in \{0, 1, \dots, q\}$,*

$$\Pr(B \le m) \le \binom{q}{m} \left(\frac{m}{q}\right)^S \le \exp\Big(q\, h(m/q) - S \log(q/m)\Big).$$

Proof. Fix a set T of m blocks. The union of these blocks has size $m\sigma = (m/q)n$. The event $\{W \subseteq \cup T\}$ has probability

$$\Pr(W \subseteq \cup T) = \frac{\binom{m\sigma}{S}}{\binom{n}{S}} \le \left(\frac{m\sigma}{n}\right)^S = \left(\frac{m}{q}\right)^S,$$

where we used $\binom{A}{S}/\binom{B}{S} \le (A/B)^S$ for $A \le B$. Union bounding over all $\binom{q}{m}$ choices of T yields the first inequality. For the second, use $\binom{q}{m} \le \exp(q h(m/q))$. □

F.3 Theorem for $t = 3$

Theorem 3 (Linear distance for $t = 3$, rate $1/2$). *Fix $\beta := 0.9$ and $\delta := 0.08$ (so $\beta \cdot (1 - H_2(\delta/\beta)) > 1/2$). There exist constants $c_0, \gamma > 0$ such that the following holds.*

Let $\sigma = \lceil c n^{1/3} \rceil$ with some constant $c \ge c_0$, rounded so that $\sigma \mid n$ and $\sigma \mid k = n/2$. Let $q = n/\sigma$. For the depth-3 ensemble $G = G_1 \pi_1 G_2 \pi_2 G_3$ defined above, for all sufficiently large n,

$$\Pr\big[d_{\min}(G) \le \delta n\big] \le \exp(-\gamma\sigma).$$

Proof. Define the random variable

$$N_{\le \delta n}(G) := \#\{x \in \mathbb{F}_2^k \setminus \{0\} : \ |xG| \le \delta n\}.$$

Then $\{d_{\min}(G) \le \delta n\} \subseteq \{N_{\le \delta n}(G) \ge 1\}$, so by Markov,

$$\Pr(d_{\min}(G) \le \delta n) \le \mathbf{E}[N_{\le \delta n}(G)].$$

We bound the expectation by splitting over the type $a = a(x)$:

$$\mathbf{E}[N_{\le \delta n}(G)] = \sum_{a=1}^{b} \sum_{x:\ a(x)=a} \Pr(|xG| \le \delta n).$$

Step 0: two deterministic observations. For any intermediate vector z partitioned into q blocks, we have

$$|z| \ge A(z) \qquad \text{and} \qquad A(z) \le |z|.$$

Moreover, for the block-diagonal maps G_2, G_3: conditional on $A(u_i) = m$, the output $v_{i+1} = u_i G_{i+1}$ consists of m independent uniform σ-bit blocks (and $q-m$ zero blocks), hence

$$|v_{i+1}| \ \big|\ (A(u_i) = m) \ \sim \ \text{Binomial}(\sigma m, 1/2). \tag{20}$$

This is exactly Lemma 3 applied blockwise, using independence of the random blocks.

For G_1: conditional on $a(x) = a$, the vector $v_1 = xG_1$ is supported on a disjoint length-2σ segments, each uniform, hence

$$S_1 := |v_1| \ \big|\ (a(x) = a) \ \sim \ \text{Binomial}(2\sigma a, 1/2). \tag{21}$$

Case I: Large types $a \ge a_\star$ with $a_\star := C\sigma$. We will choose C (a constant) later, sufficiently large.

Fix any x with $a(x) = a \ge C\sigma$. Let $S_1 = |v_1|$. By (21), $\mathbf{E}[S_1] = \sigma a$. Apply Lemma 4 with $M = 2\sigma a$ and $\rho = 1/4$:

$$\Pr\Big(S_1 \le \frac{\sigma a}{2}\Big) = \Pr\Big(S_1 \le \frac{1}{4} \cdot (2\sigma a)\Big) \le 2^{-2\sigma a\,(1 - H_2(1/4))}.$$

In particular, since $a \ge C\sigma$, the RHS is at most $\exp(-\Omega(\sigma^2))$.

Now condition on the event

$$\mathcal{E}_1 := \Big\{S_1 \ge \frac{\sigma a}{2}\Big\}.$$

Under $\mathcal{E}_1$, we have $S_1 \ge (C/2)\sigma^2$. Recall that $\sigma = cn^{1/3}$ implies $q = n/\sigma = \Theta(\sigma^2)$, more precisely $q = n/\sigma = (1/c)n^{2/3}$ and $\sigma^2 = c^2 n^{2/3} = c^3 q$. Thus $S_1 \ge (C/2)c^3 q$.

Let $B_1 := A(u_1)$ be the number of nonzero blocks after the permutation π_1. Conditional on S_1, the support of u_1 is a uniformly random S_1-subset of $[n]$. Apply Lemma 6 with $m := \lfloor \eta q \rfloor$ for some constant $\eta \in (0,1)$ (say $\eta = 0.4$):

$$\Pr\big(B_1 \le \eta q \mid S_1\big) \le \exp\big(qh(\eta) - S_1 \log(1/\eta)\big).$$

Under $\mathcal{E}_1$, $S_1 \ge \theta q$ where $\theta := (C/2)c^3$. Therefore

$$\Pr\big(B_1 \le \eta q \mid \mathcal{E}_1\big) \le \exp\big(q(h(\eta) - \theta \log(1/\eta))\big).$$

Choose C (depending only on c and η) so that $\theta > h(\eta)/\log(1/\eta)$, hence the exponent is $-\Omega(q)$. Thus, for this case,

$$\Pr(B_1 \le \eta q) \le \exp(-\Omega(q)) + \exp(-\Omega(\sigma^2)). \tag{22}$$

Next, conditional on $B_1 \ge \eta q$, the block-diagonal multiplication by G_2 yields $S_2 := |v_2|$ with $S_2 \sim \text{Binomial}(\sigma B_1, 1/2)$ by (20), and $\sigma B_1 \ge \eta\sigma q = \eta n$. Applying Lemma 4 with $M = \sigma B_1 \ge \eta n$ and $\rho = 1/4$ gives

$$\Pr\Big(S_2 \le \frac{\eta n}{4} \;\Big|\; B_1 \ge \eta q\Big) \le 2^{-\eta n\,(1 - H_2(1/4))} = \exp(-\Omega(n)). \tag{23}$$

Now condition on the event $\mathcal{E}_2 := \{S_2 \ge \alpha n\}$ where $\alpha := \eta/4$.

Under $\mathcal{E}_2$, consider $B_2 := A(u_2)$ after the permutation π_2. Again by Lemma 6, with $m := \lfloor \beta q \rfloor$ and recalling that $S_2 \ge \alpha n = \alpha\sigma q$,

$$\begin{aligned}\Pr(B_2 \le \beta q \mid S_2) &\le \exp\big(qh(\beta) - S_2 \log(1/\beta)\big) \\ &\le \exp\big(qh(\beta) - \alpha\sigma q \log(1/\beta)\big) \\ &= \exp(-\Omega(n)),\end{aligned}$$

since $\sigma \to \infty$.

Finally, conditional on $B_2 \ge \beta q$, the last layer G_3 gives $|y| = |u_2 G_3| \sim \text{Binomial}(\sigma B_2, 1/2)$ by (20), and $\sigma B_2 \ge \beta\sigma q = \beta n$. Thus $|y|$ stochastically dominates $\text{Binomial}(\beta n, 1/2)$, and

$$\Pr(|y| \le \delta n \mid B_2 \ge \beta q) \le 2^{-\beta n(1 - H_2(\delta/\beta))}. \tag{24}$$

Combining (22), (23), the bound on B_2, and (24), we obtain: for every fixed x with $a(x) \ge C\sigma$,

$$\Pr(|xG| \le \delta n) \le 2^{-\beta n(1 - H_2(\delta/\beta))} + \exp(-\Omega(n)) + \exp(-\Omega(q)).$$

Summing over all such x (at most $2^k = 2^{n/2}$ messages) gives total contribution

$$\sum_{x:\ a(x) \ge C\sigma} \Pr(|xG| \le \delta n) \le 2^{n/2} \cdot 2^{-\beta n(1 - H_2(\delta/\beta))} + \exp(-\Omega(n)) + \exp(-\Omega(q)).$$

By our choice of (β, δ), $\beta(1 - H_2(\delta/\beta)) > 1/2$, so the first term is $2^{-\Omega(n)}$. Hence the entire large-type contribution is at most $2^{-\Omega(n)} + \exp(-\Omega(q))$.

Case II: Medium types $2 \le a < C\sigma$. Fix any x with $a(x) = a$ in this range. Let $S_1 = |v_1|$. By (21) and Lemma 4 with $M = 2\sigma a$ and $\rho = 1/4$,

$$\Pr\Big(S_1 \le \frac{\sigma a}{2}\Big) \le 2^{-2\sigma a\,(1-H_2(1/4))}. \tag{25}$$

Define the event $\mathcal{F}_1 := \{S_1 \ge \sigma a/2\}$.

Now consider $B_1 = A(u_1)$. Conditional on S_1, apply Lemma 6 with

$$m := \left\lfloor \frac{S_1}{4} \right\rfloor.$$

Then

$$\Pr\Big(B_1 \le \frac{S_1}{4} \;\Big|\; S_1\Big) \le \binom{q}{m}\left(\frac{m}{q}\right)^{S_1} \le \exp\Big(qh(m/q) - S_1\log(q/m)\Big). \tag{26}$$

On $\mathcal{F}_1$, we have $S_1 \ge \sigma a/2$, and since $a \le C\sigma$ we also have $S_1 \le \sigma a + O(\sqrt{\sigma a}) \le O(\sigma^2)$. Recalling $q = \Theta(\sigma^2)$, we have $q/m = \Theta(q/S_1) = \Theta(\sigma/a)$, hence $\log(q/m) = \Omega(\log(\sigma/a))$. Thus the exponent in (26) is at most

$$-q\cdot 0 + \big(-\Omega(S_1\log(\sigma/a))\big) \le -\Omega(\sigma a\log(\sigma/a)),$$

for all sufficiently large σ. Therefore, there exists an absolute constant $c_1 > 0$ such that

$$\Pr\Big(B_1 \le \frac{S_1}{4} \;\Big|\; \mathcal{F}_1\Big) \le \exp\big(-c_1\,\sigma a\log(\sigma/a)\big). \tag{27}$$

Next, conditional on $\mathcal{F}_1$ and $B_1 \ge S_1/4$, we have $B_1 \ge S_1/4 \ge (\sigma a/8)$. Then $S_2 = |v_2|$ satisfies, by (20),

$$S_2 \;\big|\; (B_1) \;\sim\; \text{Binomial}(\sigma B_1, 1/2), \qquad \sigma B_1 \ge \sigma\cdot\frac{\sigma a}{8} = \frac{\sigma^2 a}{8}.$$

Applying Lemma 4 with $M = \sigma B_1$ and $\rho = 1/4$ gives

$$\Pr\Big(S_2 \le \frac{\sigma^2 a}{64} \;\Big|\; B_1 \ge \frac{\sigma a}{8}\Big) \le 2^{-(\sigma^2 a/8)\,(1-H_2(1/4))} = \exp(-\Omega(\sigma^2 a)). \tag{28}$$

Since $a \ge 2$, the RHS is at most $\exp(-\Omega(\sigma^2)) = \exp(-\Omega(q))$.

Now condition on $\mathcal{F}_2 := \{S_2 \ge \alpha_2 q\}$ for some constant $\alpha_2 > 0$. Indeed, (28) implies $\Pr(\neg\mathcal{F}_2) \le \exp(-\Omega(q))$ by choosing α_2 small enough, because $\sigma^2 a = \Theta(q)\cdot a \ge 2\Theta(q)$.

Under $\mathcal{F}_2$, apply Lemma 6 to π_2 with $m := \lfloor\beta_0 q\rfloor$, where we choose some constant $\beta_0 \in (0,1)$ smaller than the typical occupancy at load α_2 (for concreteness, take $\beta_0 := \alpha_2/4$; any fixed choice works). Since $S_2 \ge \alpha_2 q$, we obtain

$$\Pr(B_2 \le \beta_0 q \mid \mathcal{F}_2) \le \exp(-\Omega(q)). \tag{29}$$

Finally, conditional on $B_2 \ge \beta_0 q$, the last layer gives $|y| \sim \text{Binomial}(\sigma B_2, 1/2)$ with $\sigma B_2 \ge \beta_0\sigma q = \beta_0 n$, so by (24) (with β_0 in place of β),

$$\Pr(|y| \le \delta n \mid B_2 \ge \beta_0 q) \le 2^{-\beta_0 n(1-H_2(\delta/\beta_0))} = \exp(-\Omega(n)).$$

Putting (25), (27), (28), (29) and the final binomial tail together, we get the following bound for each fixed x with $2 \le a(x) < C\sigma$:

$$\Pr(|xG| \le \delta n) \le 2^{-\Omega(\sigma a)} + \exp\bigl(-\Omega(\sigma a \log(\sigma/a))\bigr) + \exp(-\Omega(q)) + \exp(-\Omega(n)).$$

Now sum over all such x using Lemma 5. For each fixed $a \ge 2$, the number of x with $a(x) = a$ is at most $\binom{eb}{a}^a 2^{\sigma a} \le \exp(O(a \log q) + \sigma a \log 2)$. Multiplying by the dominant per-message term $\exp(-\Omega(\sigma a \log(\sigma/a)))$ yields a total of at most $\exp(-\Omega(\sigma \log \sigma))$ for small a (and much smaller for larger a). In particular, the entire medium-type contribution satisfies

$$\sum_{x:\ 2 \le a(x) < C\sigma} \Pr(|xG| \le \delta n) \le \exp(-\Omega(\sigma \log \sigma)). \tag{30}$$

Case III: Smallest type $a = 1$. This is the regime that governs the unavoidable σ-level failure exponent.

Fix x with $a(x) = 1$. Then $S_1 = |v_1| \sim \mathrm{Binomial}(2\sigma, 1/2)$ by (21). Choose a constant $\rho := 1/8$ and define the event $\mathcal{G}_1 := \{S_1 \ge \rho\sigma\}$. By Lemma 4 with $M = 2\sigma$ and threshold $\rho\sigma = (\rho/2) \cdot (2\sigma)$,

$$\Pr(\neg\mathcal{G}_1) = \Pr\Bigl(S_1 \le \rho\sigma\Bigr) \le 2^{-2\sigma\,(1 - H_2(\rho/2))}. \tag{31}$$

With $\rho/2 = 1/16$, note that $1 - H_2(1/16) \approx 0.6627$, so the exponent is $\approx 1.325\sigma$ (strictly $> \sigma$).

Now condition on $\mathcal{G}_1$. Let $B_1 = A(u_1)$. Apply Lemma 6 with $m := \lfloor(\rho/2)\sigma\rfloor$. Since $q = \Theta(\sigma^2)$, we have $m/q = \Theta(1/\sigma)$, and since $S_1 \ge \rho\sigma$,

$$\begin{aligned}
\Pr\Bigl(B_1 \le \frac{\rho\sigma}{2} \;\Big|\; \mathcal{G}_1\Bigr) &\le \binom{q}{m}\left(\frac{m}{q}\right)^{S_1} \\
&\le \exp\Bigl(qh(m/q) - S_1 \log(q/m)\Bigr) \\
&\le \exp(-\Omega(\sigma \log \sigma)).
\end{aligned}$$

Thus, except with probability $\exp(-\Omega(\sigma \log \sigma))$, we have $B_1 \ge (\rho/2)\sigma$.

Conditional on $B_1 \ge (\rho/2)\sigma$, the second layer yields $S_2 \sim \mathrm{Binomial}(\sigma B_1, 1/2)$ with $\sigma B_1 \ge (\rho/2)\sigma^2 = \Theta(q)$. Therefore, by Lemma 4 with $\rho' = 1/4$,

$$\Pr\Bigl(S_2 \le \frac{\rho\sigma^2}{16} \;\Big|\; B_1 \ge \frac{\rho\sigma}{2}\Bigr) \le \exp(-\Omega(\sigma^2)) = \exp(-\Omega(q)).$$

On the complementary event $S_2 \ge \alpha_3 q$ for some constant $\alpha_3 > 0$, the permutation π_2 activates a constant fraction of blocks with probability $1 - \exp(-\Omega(q))$ (by Lemma 6 with constant $m = \beta_3 q$), and then the final layer yields a binomial tail on $\Theta(n)$ bits:

$$\Pr(|y| \le \delta n \mid B_2 \ge \beta_3 q) \le \exp(-\Omega(n)).$$

Collecting these bounds, we obtain for each fixed x with $a(x) = 1$:

$$\Pr(|xG| \le \delta n) \le 2^{-2\sigma(1-H_2(1/16))} + \exp(-\Omega(\sigma \log \sigma)) + \exp(-\Omega(q)) + \exp(-\Omega(n)).$$

Now sum over all x with $a(x) = 1$. There are exactly $\binom{b}{1}(2^\sigma - 1) \le b2^\sigma = (q/2)\,2^\sigma$ such messages. Hence the total type-1 contribution is at most

$$\begin{aligned}\sum_{x:\ a(x)=1} \Pr(|xG| \le \delta n) &\le \frac{q}{2}\,2^\sigma \cdot 2^{-2\sigma(1-H_2(1/16))} \ +\ \frac{q}{2}\,2^\sigma \cdot \exp(-\Omega(\sigma \log \sigma)) \\ &\quad +\ \frac{q}{2}\,2^\sigma \cdot \exp(-\Omega(q)) \\ &\le \mathrm{poly}(\sigma) \cdot 2^{-\kappa\sigma} \ +\ \exp(-\Omega(\sigma \log \sigma)) \ +\ \exp(-\Omega(q)),\end{aligned}$$

where

$$\kappa \ :=\ 2\big(1 - H_2(1/16)\big) - 1 \ >\ 0.$$

Thus there exists a constant $\gamma > 0$ such that the entire type-1 contribution is at most $\exp(-\gamma\sigma)$ for all sufficiently large σ (hence for all sufficiently large n).

Conclusion. Combining the three cases, we have

$$\mathbf{E}[N_{\le \delta n}(G)] \le \underbrace{\exp(-\gamma\sigma)}_{\text{type } a=1} \ +\ \underbrace{\exp(-\Omega(\sigma \log \sigma))}_{\text{types } 2 \le a < C\sigma} \ +\ \underbrace{2^{-\Omega(n)} + \exp(-\Omega(q))}_{\text{types } a \ge C\sigma}.$$

Since $\sigma \to \infty$ with n, the dominant term is $\exp(-\gamma\sigma)$. Therefore, for all sufficiently large n,

$$\Pr(d_{\min}(G) \le \delta n) \le \mathbf{E}[N_{\le \delta n}(G)] \le \exp(-\gamma\sigma),$$

as claimed. □

F.4 Beyond $t = 3$

The above proof shows an upper bound of the form $\exp(-\Theta(\sigma))$. This scaling is also the correct "floor" for first-moment arguments in this ensemble: there are $\Theta(q2^\sigma)$ messages with $a(x) = 1$, and for each such message x, the first-layer output v_1 is uniform in $\{0,1\}^{2\sigma}$, so events like $|v_1| = 1$ occur with probability $\Theta(\sigma) \cdot 2^{-2\sigma}$, already implying an expected $\Theta(q\sigma) \cdot 2^{-\sigma}$ population of unusually low-weight codewords (hence a natural σ-governed floor).

Remark 2 (Extension to all constant $t \ge 4$). For every constant $t \ge 4$, setting $\sigma = \lceil cn^{1/t} \rceil$ for a sufficiently large constant c yields linear minimum distance with failure probability at most $\exp(-\Omega(q))$. The proof follows the same template as above:

– For large types $a \ge C\sigma$, by time u_{t-2} the number of active blocks reaches $\Theta(q)$ with failure $\exp(-\Omega(q))$, then the last two layers yield $\exp(-\Omega(n))$ tails and the final binomial tail beats the global union bound.

– For small/medium types $a < C\sigma$, the type count is still at most $\exp(O(\sigma^2))$ while the per-message failure probability improves (in fact) to $\exp(-\Omega(\sigma^{t-1})) = \exp(-\Omega(q))$ due to the extra amplification layers.

We omit the repetition of the same Chernoff + occupancy steps, which are identical to the $t = 3$ case but easier due to the additional slack.

G Banded-Diagonal Codes

We consider an alternative formulation of our Block-Diagonal code, which we refer to as the Banded-Diagonal code. Instead of having submatrices along the diagonal, this code has a diagonal band of random values. The width of the band is σ. Asymptotically, we suspect this code performs similarly but conceptually offers some advantages.

In particular, consider a Block-Diagonal code and the string $\mathbf{x} = 0...0110...0$ which has two 1s somewhere in the string. Moreover, let us assume these two 1s fall within the same block. Then the minimum distance behavior of this $\mathbf{x}$ and that of $\mathbf{x}'$ which has only one of these 1s set is identical. That is, once a block is "on", having additional 1s in the input for this block does not change the output distribution, since in both cases it is uniform.

If we contrast this with the Banded-Diagonal construction, each additional 1 in the input does change the distribution as the non-zeros in the generator $\mathbf{G}$ do not perfectly overlap. In the worst case, $\mathbf{x}$ has two consecutive 1s, and there is then one additional output bit that will be uniform.

Next, we define the enumerator for this code. However, the resulting enumerator is more computationally expensive to compute and therefore we chose not to implement it. We leave further investigation of this approach to future work.

G.1 The Enumerator

The generator matrix $\mathbf{G} \in \{0,1\}^{k \times n}$ is of the form

$$\mathbf{G} = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \dots & \alpha_{1,\sigma} & & & \mathbf{O} \\ & \alpha_{2,1} & \alpha_{1,2} & \dots & \alpha_{1,\sigma} & & \\ & & \ddots & \ddots & & \ddots & \\ \mathbf{O} & & & \alpha_{k,1} & \alpha_{k,2} & \dots & \alpha_{k,\sigma} \end{pmatrix},$$

Each $\boldsymbol{\alpha}_i$ for $i \in [k]$ is sampled independently and uniformly at random from $\{0,1\}^{\sigma_i}$. Let

$$\boldsymbol{\alpha}_i = \left(\alpha_{i,1}\ \alpha_{i,2} \dots \alpha_{i,\sigma_i}\right),$$

where $\alpha_{i,j} \in \{0,1\}$, $i \in [k]$, $j \in [\sigma_i]$. Here, we assume that $k, n, \sigma \in \mathbb{N}$, and $k, \sigma \leq n$.

Let $\mathbf{x} \in \{0,1\}^k$ be an input word. Let

$$\mathbf{x} = \left(x_1\ x_2 \dots x_k\right),$$

where $x_i \in \{0,1\}$, $i \in [k]$. Then, the codeword $\mathbf{c} = \mathbf{xG}$ corresponding to $\mathbf{x}$ is given by

$$\mathbf{c} = \left(c_1\ c_2\ \ldots\ c_n\right),$$

where

$$c_i = \sum_{j=\max\{i-\sigma+1,1\}}^{\min\{i,k\}} x_j \alpha_{j,i+1-j},$$

$i \in [n]$.

Suppose $\mathbf{wt}_1(\mathbf{x}) = w$. Define $\mathbf{Q}_{\blacktriangle} : \{0,1\}^k \to 2^{[n]}$, where $\mathbf{Q}_{\blacktriangle}(\mathbf{x}) \subseteq [n]$ is defined as

$$\mathbf{Q}_{\blacktriangle}(\mathbf{x}) := \left\{ i \in [n] : \bigvee_{j=\max\{i-\sigma+1,1\}}^{\min\{i,k\}} (x_j = 1) \right\},$$

i.e., the set of i such that c_i is not guaranteed to be zero. Additionally, define $\mathbf{q}_{\blacktriangle} : \{0,1\}^k \to [0,n]$, where $\mathbf{q}_{\blacktriangle}(\mathbf{x}) = |\mathbf{Q}_{\blacktriangle}(\mathbf{x})|$. Also, define $\mathbf{Q} : \{0,1\}^n \to 2^{[n]}$, where $\mathbf{Q}(\mathbf{c}) \subseteq [n]$ is the set of i such that $c_i = 1$. We will abuse notation slightly to also have $\mathbf{Q} : \{0,1\}^k \to 2^{[k]}$, where $\mathbf{Q}(\mathbf{x}) \subseteq [k]$ is the set of i such that $x_i = 1$. Note that $|\mathbf{Q}(\mathbf{x})| = w$. We now have the following lemma.

Lemma 7. *For any $\mathbf{x} \in \{0,1\}^k$ and $\mathbf{y} \in \{0,1\}^n$,*

$$\Pr\left[X_{\mathbf{x},\mathbf{G},\mathbf{y}} = 1\right] = \begin{cases} 2^{-\mathbf{q}_{\blacktriangle}(\mathbf{x})} & \text{if } \mathbf{Q}(\mathbf{y}) \subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases}.$$

Proof. Clearly, if $i \notin \mathbf{Q}_{\blacktriangle}(\mathbf{x})$, then $c_i = 0$. Thus, $\mathbf{Q}(\mathbf{c}) \subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x})$. In other words, if $\mathbf{Q}(\mathbf{y}) \not\subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x})$, then $\Pr\left[X_{\mathbf{x},\mathbf{G},\mathbf{y}} = 1\right] = 0$.

Consider any $\mathbf{x} \in \{0,1\}^k$ and $\mathbf{y} \in \{0,1\}^n$ such that $\mathbf{Q}(\mathbf{y}) \subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x})$. Let

$$\mathbf{y} = \left(y_1\ y_2\ \ldots\ y_n\right),$$

where $y_i \in \{0,1\}$, $i \in [n]$. Consider $i \in \mathbf{Q}_{\blacktriangle}(\mathbf{x})$, i.e., an i such that $x_j = 1$ for some $j \in [\max\{i-\sigma+1,1\}, \min\{i,k\}]$. Then,

$$\Pr\left[y_i = \sum_{j=\max\{i-\sigma+1,1\}}^{\min\{i,k\}} x_j \alpha_{j,i+1-j} \right] = \frac{1}{2}.$$

Therefore,

$$\Pr\left[\forall i \in \mathbf{Q}_{\blacktriangle}(\mathbf{x}) : y_i = \sum_{j=\max\{i-\sigma+1,1\}}^{\min\{i,k\}} x_j \alpha_{j,i+1-j} \right] = 2^{-\mathbf{q}_{\blacktriangle}(\mathbf{x})}.$$

Since $\mathbf{Q}(\mathbf{y}) \subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x})$,

$$\Pr\left[\forall i \notin \mathbf{Q}_{\blacktriangle}(\mathbf{x}) : y_i = \sum_{j=\max\{i-\sigma+1,1\}}^{\min\{i,k\}} x_j \alpha_{j,i+1-j} \right] = 1.$$

Therefore,

$$\Pr\left[X_{\mathbf{x},\mathbf{G},\mathbf{y}} = 1\right] = 2^{-\mathbf{q}_{\blacktriangle}(\mathbf{x})}.$$

□

We now have

$$\mathbb{E}_{\mathbf{G}\leftarrow\mathcal{D}}\left[E^{\mathbf{G}}_{w,h}\right] = \sum_{\substack{\mathbf{x}\in\{0,1\}^k,\mathbf{y}\in\{0,1\}^n \\ \mathbf{Q}(\mathbf{y})\subseteq\mathbf{Q}_{\blacktriangle}(\mathbf{x}) \\ \mathbf{wt}_1(\mathbf{x})=w,\mathbf{wt}_1(\mathbf{y})=h}} 2^{-\mathbf{q}_{\blacktriangle}(\mathbf{x})}, \tag{32}$$

i.e., we need to count the number of $\mathbf{x} \in \{0,1\}^k$, $\mathbf{y} \in \{0,1\}^n$ with $\mathbf{Q}(\mathbf{y}) \subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x})$, $\mathbf{wt}_1(\mathbf{x}) = w$, and $\mathbf{wt}_1(\mathbf{y}) = h$. We perform this count using two intermediate variables. The first is $q = \mathbf{q}_{\blacktriangle}(\mathbf{x})$. $q \in [w, n]$, w being the worst case, when all the 1s in $\mathbf{x}$ appear at the tail end. The second is r which denotes the number of *runs* in $\mathbf{Q}_{\blacktriangle}(\mathbf{x})$. To define runs, we begin by alternatively defining $\mathbf{Q}_{\blacktriangle}(\mathbf{x})$. For $i \in [k]$, let $I_i = [i, \min\{i + \sigma - 1, n\}]$. It is immediate to see that

$$\mathbf{Q}_{\blacktriangle}(\mathbf{x}) = \bigcup_{i\in\mathbf{Q}(\mathbf{x})} I_i.$$

We use the way I_is, for $i \in \mathbf{Q}(\mathbf{x})$, overlap to define the runs in $\mathbf{Q}_{\blacktriangle}(\mathbf{x})$. Define

$$\mathbf{first}(\mathbf{x}) := \min_{t\in\mathbf{Q}(\mathbf{x})} t.$$

Note that if there exists no $t \in \mathbf{Q}(\mathbf{x})$ (i.e., if $\mathbf{x} = 0^k$), then $\mathbf{first}(\mathbf{x}) := 0$. For any $i \in [n]$, define

$$\mathbf{next}^{\Delta}_{\mathbf{x}}(i) := \min_{\substack{t\in\mathbb{N} \\ i+t\in\mathbf{Q}(\mathbf{x})}} t.$$

Note that if for any $i \in [n]$ there exists no $t \in \mathbb{N}$ such that $i + t \in \mathbf{Q}(\mathbf{x})$, then $\mathbf{next}^{\Delta}_{\mathbf{x}}(i) := \infty$. For any $i \in [n]$, define

$$\mathbf{next}_{\mathbf{x}}(i) := i + \mathbf{next}^{\Delta}_{\mathbf{x}}(i).$$

Define $\mathbf{R} : \{0,1\}^k \to 2^{2^{[n]}}$, where $\mathbf{R}(\mathbf{x}) \subseteq 2^{[n]}$, constructively as follows:

1. Let $i = \mathbf{first}(\mathbf{x})$. If $i = 0$, return $\mathbf{R}(\mathbf{x}) = \emptyset$.
2. Set $\mathbf{r} := \{i\}$, $r = 1$, and $j := i$.
3. `//` I_j `and` $I_{\mathbf{next}_{\mathbf{x}}(j)}$ `do not overlap, so start a new run`
   If $\mathbf{next}^{\Delta}_{\mathbf{x}}(j) \geq \sigma - 1$, add $R_r = \bigcup_{t\in\mathbf{r}} I_t$ to $\mathbf{R}(\mathbf{x})$ and set $i := \mathbf{next}_{\mathbf{x}}(j)$ and $r = r + 1$. Return $\mathbf{R}(\mathbf{x})$ if $i = \infty$, otherwise, go back to Step 2.
   `//` I_j `and` $I_{\mathbf{next}_{\mathbf{x}}(j)}$ `overlap, so continue the same run`
   If $\mathbf{next}^{\Delta}_{\mathbf{x}}(j) < \sigma - 1$, add $\mathbf{next}_{\mathbf{x}}(j)$ to $\mathbf{r}$, set $j := \mathbf{next}_{\mathbf{x}}(j)$, and go back to Step 3.

Essentially, a run is a maximal set of contiguous indices in $\mathbf{Q}_{\blacktriangle}(\mathbf{x})$ belonging to overlapping I_is for $i \in \mathbf{Q}(\mathbf{x})$, and $\mathbf{R}(\mathbf{x})$ is the set of runs in $\mathbf{Q}_{\blacktriangle}(\mathbf{x})$. The intermediate variable r is defined as $r = |\mathbf{R}(\mathbf{x})|$. Since $|\mathbf{x}| = w$, $r \in [0, w]$ by definition.

We now compute the number of $\mathbf{x} \in \{0,1\}^k$ such that $\mathbf{wt}_1(\mathbf{x}) = w$, $\mathbf{q}_{\blacktriangle}(\mathbf{x}) = q$, and $|\mathbf{R}(\mathbf{x})| = r$ in two cases.

Case 1: $q, r, w > 0$.

Let $\mathbf{R}(\mathbf{x}) = \{R_1, \ldots, R_r\}$ as determined by the algorithm above for constructing $\mathbf{R}(\mathbf{x})$, where $R_i \subseteq [n]$ for $i \in [r]$. By definition, $\mathbf{x}_{R_r}$ must be of the form

$$\mathbf{x}_{R_r} = \mathbf{u}_r 10^c,$$

for some $\mathbf{u}_r \in \{0,1\}^*$ and $\alpha \in [0, \sigma - 1]$. Furthermore, for $i \in [r-1]$, $\mathbf{x}_{R_i}$ must be of the form

$$\mathbf{x}_{R_i} = \mathbf{u}_i 10^{\sigma-1},$$

for some $\mathbf{u}_i \in \{0,1\}^*$. We call a run R, complete, if $\mathbf{x}_R$ is of the form $\mathbf{x}_R = \mathbf{u}10^{\sigma-1}$ for some $\mathbf{u} \in \{0,1\}^*$. Similarly, we call it α-incomplete, if $\mathbf{x}_R$ is of the form $\mathbf{x}_R = \mathbf{u}10^{\sigma-1-\alpha}$, for some $\mathbf{u} \in \{0,1\}^*$ and $\alpha \in [\sigma-1]$. Our prior observation can now be restated as for $i \in [r-1]$, R_i is complete, while R_r may either be complete or α-incomplete for some $\alpha \in [\sigma - 1]$.

We next consider $\mathbf{u}_i$ for $i \in [r]$. By definition, each $\mathbf{u}_i$ for $i \in [r]$ must be of the form

$$\mathbf{u}_i = 10^{\ell_{i,1}} \ldots 10^{\ell_{i,t_i}}$$

where $t_i \in \mathbb{N} \cup \{0\}$, and $\ell_{i,j} \in [0, \sigma - 2]$ for all $j \in [t_i]$. We call the (ordered) collection of strings $10^{\ell_{i,j}}$ for $i \in [r]$ and $j \in [t_i]$ the set of mini-runs of $\mathbf{x}$. By definition, the number of mini-runs is given by

$$\sum_{i \in [r]} t_i = w - r.$$

Also,

$$\tilde{q} := \sum_{i \in [r]} |\mathbf{x}_{R_i}| = q - \min\{\sigma - 1 - \alpha, n - k\},$$

and

$$\sum_{i \in [r]} \mathbf{wt}_1(\mathbf{x}_{R_i}) = \mathbf{wt}_1(\mathbf{x}) = w.$$

Therefore,

$$\sum_{i \in [r]} \mathbf{wt}_0(\mathbf{x}_{R_i}) = \tilde{q} - w,$$

and

$$\begin{aligned} \sum_{i \in [r]} \mathbf{wt}_0(\mathbf{u}_i) &= \sum_{i \in [r]} \mathbf{wt}_0(\mathbf{x}_{R_i}) - (r-1)(\sigma - 1) - \alpha \\ &= \tilde{q} - w - (r-1)(\sigma - 1) - \alpha. \end{aligned}$$

Finally,

$$\mathbf{wt}_0(\mathbf{x}_{[k] \setminus \mathbf{Q}_{\blacktriangle}(\mathbf{x})}) = n - \tilde{q}$$

Now, enumerating the $\mathbf{x} \in \{0,1\}^k$ such that $\mathbf{wt}_1(\mathbf{x}) = w$, $\mathbf{q}_{\blacktriangle}(\mathbf{x}) = q$, and $|\mathbf{R}(\mathbf{x})| = r$ can be broken down into the following four steps:

(1) Enumerate over $\alpha \in [0, \sigma - 1]$, i.e., over whether the rth run is complete or α-incomplete.

(2) For each α, enumerate over the possible sets of mini-runs of $\mathbf{x}$.

(3) For each set of mini-runs of $\mathbf{x}$, enumerate over the possible assignments of the mini-runs to each of the r runs.

(4) For each assignment of the mini-runs of $\mathbf{x}$ to each of the r runs, enumerate over the possible assignments 0s from $\mathbf{x}_{[k]\setminus \mathbf{Q}_{\blacktriangle}(\mathbf{x})}$ to the regions surrounding each of the r runs.

We proceed to work through each step. For each $\alpha \in [0, \sigma - 1]$, enumerating over the possible sets of mini-runs of $\mathbf{x}$ is equivalent to the number of ways in which we can assign $\sum_{i\in[r]} \mathbf{wt}_0(\mathbf{u}_i)$ identical balls into $\sum_{i\in[r]} t_i$ labeled bins, each filled with at most $\sigma - 2$ balls. The number of ways to do this is thus

$$\begin{aligned} E^{(2),c}_{w,q,r} &:= N\left(\sum_{i\in[r]} \mathbf{wt}_0(\mathbf{u}_i), \sum_{i\in[r]} t_i, \sigma - 2\right) \\ &= \sum_{i=0}^{w-r} (-1)^i \binom{w-r}{i} \binom{\tilde{q} - \sigma(r+i-1) + i - \alpha - 2}{w-r-1}. \end{aligned}$$

For each set of mini-runs of $\mathbf{x}$, enumerating over the possible assignments of the mini-runs to each of the r runs is equivalent to the number of solutions to $\sum_{i\in[r]} t_i = w - r$ where $t_i \in \mathbb{N} \cup \{0\}$ for $i \in [r]$. This is equal to

$$E^{(3),c}_{w,q,r} := \mathcal{B}(w-r, r) = \binom{w-1}{r-1}.$$

For each assignment of the mini-runs of $\mathbf{x}$ to each of the r runs, enumerating over the possible assignments 0s from $\mathbf{x}_{[k]\setminus \mathbf{Q}_{\blacktriangle}(\mathbf{x})}$ to the regions surrounding each of the r runs is equivalent to the number of ways in which we can assign $\mathbf{wt}_0(\mathbf{x}_{[k]\setminus \mathbf{Q}_{\blacktriangle}(\mathbf{x})})$ identical balls into $r+1$ labeled bins, if $c = \sigma - 1$ (i.e., if the rth run is complete), and into r labeled bins, if $c < \sigma - 1$ (i.e., if the rth run is α-incomplete for some $\alpha \in [\sigma - 1]$), since 0s cannot follow an incomplete run. The number of ways to do this is thus

$$\begin{aligned} E^{(4),c}_{w,q,r} &:= \begin{cases} \mathcal{B}(\mathbf{wt}_0(\mathbf{x}_{[k]\setminus \mathbf{Q}_{\blacktriangle}(\mathbf{x})}), r+1) & \text{if } c = \sigma - 1 \\ \mathcal{B}(\mathbf{wt}_0(\mathbf{x}_{[k]\setminus \mathbf{Q}_{\blacktriangle}(\mathbf{x})}), r) & \text{otherwise} \end{cases} \\ &= \begin{cases} \binom{n-\tilde{q}+r}{r} & \text{if } c = \sigma - 1 \\ \binom{n-\tilde{q}+r-1}{r-1} & \text{otherwise} \end{cases}. \end{aligned}$$

Putting the four steps together, the number of $\mathbf{x} \in \{0,1\}^k$ such that $\mathbf{wt}_1(\mathbf{x}) = w$, $\mathbf{q}_{\blacktriangle}(\mathbf{x}) = q$, and $|\mathbf{R}(\mathbf{x})| = r$ is

$$\begin{aligned}
E_{w,q,r} &:= \sum_{c=0}^{\sigma-1} E^{(2),c}_{w,q,r} \cdot E^{(3),c}_{w,q,r} \cdot E^{(4),c}_{w,q,r} \\
&= \binom{w-1}{r-1} \cdot \left[\sum_{i=0}^{w-r} (-1)^i \binom{w-r}{i} \binom{n-\tilde{q}+r}{r} \binom{\tilde{q}-\sigma(r+i)+i-1}{w-r-1} \right. \\
&\left. + \sum_{c=0}^{\sigma-2} \sum_{i=0}^{w-r} (-1)^i \binom{w-r}{i} \binom{n-\tilde{q}+r-1}{r-1} \binom{\tilde{q}-\sigma(r+i-1)+i-c-2}{w-r-1} \right]
\end{aligned}$$

where $\tilde{q} = q - \min\{\sigma - 1 - c, n - k\}$.

Case 2: We now consider the cases where at least one of q, r, w is 0. Here, note that $E_{0,0,0} := 1$ but at least one of q, r, w are > 0 while at least one of them is 0, then $E_{w,q,r} := 0$.

Now, consider any $\mathbf{x} \in \{0,1\}^k$ such that $\mathbf{wt}_1(\mathbf{x}) = w$, $\mathbf{q}_{\blacktriangle}(\mathbf{x}) = q$, and $|\mathbf{R}(\mathbf{x})| = r$. We will count the number of $\mathbf{y} \in \{0,1\}^n$ such that $\mathbf{wt}_1(\mathbf{y}) = h$ and $\mathbf{Q}(\mathbf{y}) \subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x})$. This is simply equal to the number of ways of choosing h bits among the q bits of y_i for $i \in \mathbf{Q}_{\blacktriangle}(\mathbf{x})$ to set to 1. The number of ways to do this is thus

$$\mathbb{E}_{q,r,h} := \binom{q}{h}.$$

Note that $\mathbb{E}_{q,r,h}$ does not depend on r. Putting everything together, we have

$$\begin{aligned}
\mathbb{E}_{\mathbf{G} \leftarrow \mathcal{D}} \left[E^{\mathbf{G}}_{w,h} \right] &\sum_{\substack{\mathbf{x} \in \{0,1\}^k, \mathbf{y} \in \{0,1\}^n \\ \mathbf{Q}(\mathbf{y}) \subseteq \mathbf{Q}_{\blacktriangle}(\mathbf{x}) \\ \mathbf{wt}_1(\mathbf{x}) = w, \mathbf{wt}_1(\mathbf{y}) = h}} 2^{-\mathbf{q}_{\blacktriangle}(\mathbf{x})} \\
&= \sum_{q=w}^{n} \sum_{r=0}^{w} 2^{-q} \cdot \mathbb{E}_{w,q,r} \cdot \mathbb{E}_{q,r,h} \\
&= \sum_{q=w}^{n} \sum_{r=0}^{w} \left[2^{-q} \cdot \binom{q}{h} \cdot E_{w,q,r} \right].
\end{aligned}$$

Disclaimer

Case studies, comparisons, statistics, research and recommendations are provided "AS IS" and intended for informational purposes only and should not be relied upon for operational, marketing, legal, technical, tax, financial or other advice. Visa Inc. neither makes any warranty or representation as to the completeness or accuracy of the information within this document, nor assumes any liability or responsibility that may result from reliance on such information. The Information contained herein is not intended as investment or legal advice, and readers are encouraged to seek the advice of a competent professional where such advice is required.

These materials and best practice recommendations are provided for informational purposes only and should not be relied upon for marketing, legal, regulatory or other advice. Recommended marketing materials should be independently evaluated in light of your specific business needs and any applicable laws and regulations. Visa is not responsible for your use of the marketing materials, best practice recommendations, or other information, including errors of any kind, contained in this document.