THEORY OF COMPUTING, Volume 16 (18), 2020, pp. 1–68
www.theoryofcomputing.org

Relaxed Locally Correctable Codes

Tom Gur* Govind Ramnarayan† Ron D. Rothblum‡

Received December 2, 2017; Revised July 31, 2020; Published December 30, 2020

Abstract. Locally decodable codes (LDCs) and locally correctable codes (LCCs) are error-correcting codes in which individual bits of the message and codeword, respectively, can be recovered by querying only a few bits from a noisy codeword. These codes have found numerous applications both in theory and in practice.

A natural relaxation of LDCs, introduced by Ben-Sasson *et al.* (SICOMP, 2006), allows the decoder to reject (i. e., refuse to answer) in case it detects that the codeword is corrupt. They call such a decoder a *relaxed decoder* and construct a constant-query relaxed LDC with nearly linear block length, whereas the state-of-the-art (full-fledged) LDCs in the constant-query regime have superpolynomial block length.

We consider an analogous relaxation for local *correction*. Thus, a *relaxed local corrector* reads only few bits from a (possibly) corrupt codeword and either recovers the desired bit of the codeword, or rejects in case it detects corruption.

We give constructions of relaxed LCCs in two regimes, where the first optimizes the query complexity and the second optimizes the rate.

An extended abstract of this paper appeared in the Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS 2018).

*Supported by the UKRI Future Leaders Fellowship.

†Parts of this research were done while the author was supported by NSF grant number 1218547. Supported by Army Research Office grant number W911NF1910217.

‡Parts of this research were done while the author was at MIT and Northeastern University. Partially supported by NSF MACS - CNS-1413920, Simons Investigator award Agreement Dated 6-5-12, the Cybersecurity and Privacy Institute at Northeastern University and a Milgrom family grant.

ACM Classification: Theory of Computation: Error-correcting codes

AMS Classification: 68R01, 68R99

Key words and phrases: codes, PCPs, locally correctable codes

 DOI: 10.4086/toc.2020.v016a018

1. **Constant Query Complexity:** A relaxed LCC with polynomial block length whose corrector only reads a constant number of bits of the codeword. In contrast, the best upper bound on the block length achieved by known q-query (full-fledged) LCCs, for constant q, is $\exp(O(n^{1/(q-1)}))$.
2. **Constant Rate:** A relaxed LCC with *constant rate* (i. e., linear block length) with quasi-polylogarithmic query complexity (specifically, $\exp(O(\log\log n)^2))$. In contrast, the best known upper bound on the query complexity of constant-rate (full-fledged) LCCs is $\exp(\widetilde{O}(\sqrt{\log n}))$ (Kopparty *et al.*, STOC'16). Our construction, however, is based on a *locally testable code* constructed by Kopparty *et al.*, which we show to allow for *relaxed* local correction with better parameters.

Contents

1 Introduction

Dating back to the seminal work of Shannon [47] and Hamming [34], error correcting codes are used to reliably transmit data over noisy channels and store data. Roughly speaking, error correcting codes are injective functions that take a message and output a codeword, in which the message is encoded with extra redundancy, with the property that even if some of the symbols in the codeword are corrupted, the message is still recoverable. Some of the main parameters of codes are their *block length*, which is the length of a codeword as a function of n, the message length; the *rate*, which is the message length divided by the block length; and the *relative distance*, which is the minimal relative Hamming distance of every pair of distinct codewords.

Locally decodable codes (LDCs) and *locally correctable codes* (LCCs) are error correcting codes that admit highly efficient procedures for recovering small amounts of data. More specifically, in an LDC, a single symbol of the message can be recovered by only reading a few bits from a noisy codeword. An LCC has the same property but with respect to recovering bits of the *codeword* itself (rather than the message).

Locally decodable codes and locally correctable codes have had a profound impact in various areas of theoretical computer science including cryptography, PCPs, hardness of approximation, interactive proofs, private information retrieval, program checking, and databases. (See [54] and the more recent [38] for surveys on locally decodable and correctable codes.) While these codes have found numerous uses in theory and practice, one significant downside is that current constructions require adding a large amount of redundancy. Specifically, to decode or correct with a *constant* number of queries, the current state-of-the-art LDCs have superpolynomial block length (Yekhanin [53], Efremenko [17]) and the best upper bound on the block length achieved by known constant-query LCCs is[1] $\exp(c_q(n^{1/(q-1)}))$ where $c_q \sim q \log_2 q$ for q queries.

Motivated by this, Ben-Sasson *et al.* [7] defined a natural relaxation of local decoding, for which they could achieve a dramatically better block length. Roughly speaking, their relaxation allows the decoder to abort in case of failure, while still requiring the decoder to successfully decode non-corrupted codewords (in particular, this prevents the decoder from always aborting). Moreover, in the constant query regime, such codes can be transformed to codes, with similar parameters, that are guaranteed to successfully decode on a constant fraction of message bits.

Thus, a *relaxed local decoder* for a code C gets oracle access to a string w that is relatively close to some codeword $c = C(x)$ and an index $i \in [|x|]$. The decoder should make only few queries to w and satisfy the following:

1. If the string $w = c$ (i. e., w is an uncorrupted codeword), the relaxed decoder must always output x_i.

2. Otherwise, with high probability, the decoder should either output x_i or a special "abort" symbol $\perp$ (indicating the decoder detected an error and is unable to decode).[2]

[1] These are Reed–Muller codes over a constant-size alphabet and with constant degree (but large dimension).

[2] The actual definition in [7] also requires that for a constant fraction of the coordinates, the decoder decodes correctly (i. e., does not output $\perp$) with high probability. However, they later show that this additional condition follows from Conditions (1) and (2) above when the decoder has $O(1)$ query complexity. See further discussion in Remark 1.3.

The additional freedom introduced by allowing the decoder to abort turns out to be extremely useful. Using the notion of PCPs of proximity (PCPP), which they also introduce[3] and construct, Ben-Sasson *et al.* obtain relaxed locally decodable codes (RLDCs) with constant query complexity and nearly linear block length.

In this article we extend the relaxation of Ben-Sasson *et al.* to LCCs and define the analogous notion of relaxed LCCs as follows: We say that a code $C : \Sigma^k \to \Sigma^n$ is a *relaxed* LCC with query complexity q, if there exists a corrector that has oracle access to a string $w \in \Sigma^n$, which is close to some codeword $c \in C$, and also gets as explicit input an index $i \in [n]$. The algorithm makes at most q queries to the string w, and satisfies the following:

1. If $w = c$ (i. e., w was not corrupted), then the corrector always outputs c_i.
2. Otherwise, with high probability, the corrector either outputs c_i, or a special "abort" symbol $\perp$.

The remarkable savings achieved by Ben-Sasson *et al.* begs the question: can relaxed locally *correctable* codes achieve similar savings in block length over current constructions of locally correctable codes? We answer this question in the affirmative by constructing relaxed LCCs with significantly better parameters than that of the state-of-the-art (full-fledged) LCCs.

1.1 Our results

In this paper we construct relaxed locally correctable codes in two different parameter regimes: the first, which we view as our main technical contribution, focuses on the constant *query complexity* regime, whereas the second, which is easier to prove given previous work, focuses on constant *rate*.

Constant-query RLCC. Our first result is a relaxed LCC which requires only $O(1)$ queries and has a polynomial block length.

Theorem 1.1 (Constant-query relaxed LCC, informally stated)**.** *There exists a relaxed* LCC $C : \{0,1\}^k \to \{0,1\}^n$ *with constant relative distance, constant query complexity, and block length* $n = \mathrm{poly}(k)$. *Furthermore, C is a linear code.*

Theorem 1.1 shows that in the constant-query regime, relaxed LCCs can achieve polynomial block length, while the upper bound on the block length achieved by the best known (full-fledged) LCCs with constant query complexity is $\exp(\exp(\widetilde{O}((\log n)^{\varepsilon})))$ where $\varepsilon = 1/\lfloor \log_2 q \rfloor$ for q queries [17].

The proof of Theorem 1.1 heavily relies on a certain type of PCPs of proximity (PCPPs) that we construct. We elaborate on our PCPP constructions in Section 1.1.1 below.

We remark that the specific block length in Theorem 1.1 is roughly *quartic* (i. e., fourth power) in the message length. Constructing a constant-query RLCC with a shorter block length (let alone an (nearly) linear one, as known for relaxed LDCs) is an interesting open problem.[4]

[3] The equivalent notion of *assignment tester* was introduced independently by Dinur and Reingold [16].

[4] Following the initial publication of this work, the aforementioned open problem was recently resolved by Chiesa, Gur, and Shinkar [12]. See Section 1.3 for details.

Constant-rate RLCC. Our second main result is a construction of a relaxed LCC with *constant rate* and quasi-polylogarithmic query complexity.

Theorem 1.2 (Constant-rate relaxed LCC, informally stated). *There exists a relaxed* LCC $C : \{0,1\}^k \to \{0,1\}^n$ *with constant relative distance, query complexity* $\exp(O(\log\log n)^2)$*, and constant rate (i. e., block length* $n = O(k)$*). Furthermore, C is a linear code and has distance–rate tradeoff approaching the Zyablov bound.*[5]

This result shows that in the constant-rate regime, there exist relaxed LCCs with query complexity $\exp(O(\log\log n)^2)$, whereas the upper bound on the number of queries in the state-of-the-art construction of constant-rate (full-fledged) LCCs, due to Kopparty *et al.* [37], is $\exp(O(\sqrt{\log n \cdot \log\log n}))$.

Actually, our construction is essentially identical to one of the constructions of [37].[6] Our main insight in proving Theorem 1.2 is that their code allows for *relaxed* local correction with much better parameters.[7] Just like in [37], as an intermediate result we first construct a large-alphabet code with the same asymptotic query complexity that approaches the Singleton bound, which may also be of interest (see Theorem 5.2). As a secondary contribution, we also provide a modular presentation for the *distance amplification* step, which is the main step in [37] (and is originally due to Alon, Edmonds and Luby [1]).

Remark 1.3. As mentioned in Footnote 2, the original definition of RLDC [7] includes a third condition, which requires that the decoder must successfully decode a constant fraction of the coordinates. More precisely, for every $w \in \Sigma^n$ that is close to some codeword $c = C(x)$, there exists a set $I_w \subseteq [k]$ of size $\Omega(k)$ such that for every $i \in I_w$ with high probability the decoder D outputs x_i (rather than outputting $\perp$).

Ben-Sasson *et al.* showed that every RLDC with constant query complexity that satisfies the first two conditions, can be transformed into an RLDC with similar parameters that satisfies the third condition as well. We remark that this transformation also applies to RLCCs with constant query complexity. However, for superconstant query complexity (as in Theorem 1.2) the same transformation only guarantees successful decoding of a constant fraction of coordinates, if the codeword is corrupted on a subconstant fraction of its coordinates (i. e., the fraction roughly corresponds to the reciprocal of the query complexity).

Remark 1.4. Both our constant-query and constant-rate RLCCs are systematic.[8] Hence they are automatically also relaxed locally *decodable* codes (i. e., RLDCs). In particular, the code from Theorem 1.2 is also the first construction of a relaxed locally *decodable* code in the constant-rate regime, with query complexity $\exp(O(\log\log n)^2)$.

1.1.1 PCP constructions

PCPs of proximity (PCPP), first studied by Ben-Sasson *et al.* [7] and by Dinur and Reingold [16] were originally introduced to facilitate PCP composition. Beyond their usefulness in PCP constructions, PCPPs have proved to be extremely useful in coding theory as well. Indeed, PCPPs lie at the heart

[5] For the *Zyablov bound* [55], see [32, Chap. 10.2].

[6] Interestingly, our construction is inspired by the [37] construction of a locally *testable* code, rather than their locally *correctable* code.

[7] We were informed that a similar observation about the [37] code has been made recently and independently in an unpublished paper by Hemenway, Ron-Zewi, and Wootters [35].

[8] Recall that a code is *systematic* if the first part of every codeword is the original message.

of several constructions of LTCs [27], relaxed LDCs [7, 26], universal LTCs [25, 24], as well as in our construction of relaxed LCCs (specifically in Theorem 1.1).

Loosely speaking, a PCPP is a proof system that allows for probabilistic verification of approximate decision problems by querying only a small number of locations in both the statement and the proof. (In contrast, a standard PCP verifier reads the *entire* statement, and probabilistically verifies an *exact* decision problem, by querying only a small number of locations in the proof.) Similarly to the scenario in property testing, the soundness guarantee provided by PCPPs is that the PCPP verifier is only required to reject statements that are "far" (in Hamming distance) from being correct.

In this paper we provide new constructions of PCPPs that play a crucial role in our constant-query relaxed LCC construction. The PCPPs that we construct are for verifying membership in affine subspaces (rather than general languages in P or NP), since this is all that we need for our RLCC constructions. More specifically, we shall construct PCPPs that are: *linear*, *robust*, *self-correctable*, and admit *strongly canonical soundness*. We discuss these properties in more detail next (see Section 4.2.5 for precise definitions). We remark that our PCPP construction is inspired by the construction of linear-inner proof-systems (LIPS) by Goldreich and Sudan [27].

Linearity. We call a PCPP proof-system *linear* if it satisfies two conditions. First, the prescribed proof π for any statement x must be a linear function of the statement. Second, to decide whether to accept, the PCPP verifier only checks that the values that it reads from the input and PCPP proof lie in an affine subspace. Put differently, the verifier's decision predicate is computable by a conjunction of linear functions. We remark that in the literature [8, 41], the term "linear PCPP" sometimes refers only to the latter of the two requirements but here we also insist that the proof be a linear function of the statement.

We use linearity both to assure that our resulting codes are linear codes, as well as to facilitate composition with other PCPPs. We note that standard PCPs are typically inherently non-linear (since they are designed for general languages in P or in NP). However, in our context we are only trying to verify membership in affine spaces and so it is reasonable to expect to have linear PCPPs.

Robustness. The notion of *robust* PCPPs, introduced by Ben Sasson *et al.* [7], refers to PCPP systems whose verifier, roughly speaking, is not only required to reject statements that are far from valid but also that the local view of the verifier (i. e., the answers to the queries made by verifier) is far from any local view that would have caused the verifier to accept. Robustness plays a key role in enabling PCP composition. While this condition holds trivially for verifiers with constant query complexity, in our construction we will also consider verifiers with superconstant query complexity, for which achieving robustness is non-trivial.

Self-correctability. In a *self-correctable* PCPP system, the proof oracle admits a local correction procedure that allows for local recovery of individual bits of a moderately corrupted PCPP proof. In other words, the proof string itself is a locally correctable code. The self-correctability of the PCPP oracles allows us to include them as part of an RLCC's codeword.

Strongly canonical soundness. The notion of PCPPs with *strongly canonical soundness*, introduced by Goldreich and Sudan [27], requires that correct inputs (i. e., that reside in the target language) have a canonical proof and the PCPP verifier is required to reject "wrong" (i. e., non-canonical) proofs, *even for correct statements*. In more detail, these PCPPs satisfy two additional requirements: (1) *canonicity*: for every true statement there exists a unique canonical proof that the verifier is required to always accept, and (2) *strongly canonical soundness*: the verifier is required to reject any pair (x, π) of statement and proof with probability that is roughly proportional to its distance from a true statement and its corresponding canonical proof.

We are now ready to state our results on PCPs of proximity with the aforementioned properties. The first construction has exponential length and constant query complexity, whereas the second construction, whose proof is significantly more involved, has polynomial length and polylogarithmic query complexity.

Our first result is a variant of the Hadamard PCPP, with exponential length but constant query complexity.

Theorem 1.5 (Informally stated, see Theorem 4.26)**.** *There exists a linear, self-correctable, strongly canonical* PCPP *for membership in affine subspaces, with query complexity* $O(1)$ *and exponential length (in the size of the statement).*

Our second result is a variant of the [5] PCP, which has polylogarithmic query complexity and polynomial length.

Theorem 1.6 (Informally stated, see Theorem 4.27)**.** *There exists a linear, self-correctable,* $\Omega(1)$*-robust, strongly canonical* PCPP *for membership in affine subspaces, with query complexity* $\mathrm{polylog}(n)$ *and* $\mathrm{poly}(n)$ *length, for statements of length n.*

1.2 Technical overview

The techniques used for our two constructions are quite different. We first outline the constant-query result, which is more complex, in Section 1.2.1 and then outline the constant-rate result in Section 1.2.2.

1.2.1 Constant-query relaxed LCC

The starting point for our construction is the [7] construction of relaxed locally *decodable* codes (RLDC), which we review next.[9] In their construction, each codeword has two parts: the first part provides the distance, and the second enables relaxed local decodability. More specifically, they construct an RLDC C' whose codewords consist of the following two equal-length parts: (1) repetitions of a codeword $C(x)$, where $C : \{0,1\}^k \to \{0,1\}^n$ is some systematic code with constant distance and rate, (2) for every message bit in $C(x)$, they add a PCPP, which is a proof that x_i is indeed the i^{th} bit of $C(x)$.[10]

We remark that the repetitions in the first part of the code are simply meant to ensure distance (as the PCPP proof strings are not necessarily a code with good distance). To decode, the relaxed decoder for C' invokes the PCPP verifier to check that the i'th bit of the first part is indeed $C(x)_i$ and outputs it, unless the verifier rejects, in which case the relaxed decoder may return $\perp$.

[9] We describe the simpler variant of the [7] construction, which achieves nearly *quadratic* block length. We remark that [7] also present a more involved construction that achieves nearly *linear* block length.

[10] Actually, our presentation differs slightly from that of [7]. Their construction contains an additional part that consists of repetitions of the original *message*. However, when using a systematic code C, this addition is not necessary.

Ben-Sasson *et al.* show that this code is indeed a relaxed LDC. However, in general, it will not necessarily be a relaxed LCC. Specifically, it is unclear how to correct bits that are part of the PCPP proof strings. Simply appending even more PCPPs to deal with the original ones will not do since we will also need to correct those. Moreover, it is worth pointing out that even if the PCPP proof strings had some internal self-correction mechanism, this would still not suffice since each PCPP proof string by itself is very short (as compared to the entire codeword) and could therefore be entirely corrupted.

Before proceeding to cope with this difficulty, we first suggest a different perspective on the [7] construction, which is inspired by the highly influential and useful notion of PCP *composition* [3]. Specifically, we think of the [7] construction as a *composition* of the code C, which is trivially locally decodable with n queries, with a constant-query PCPP. This composition yields a *relaxed* LDC with query complexity $O(1)$, at a moderate increase in block length (which comes from appending all of the PCPP proof strings).

We shall adopt the composition perspective, and use it to construct a relaxed locally *correctable* code, by introducing a technique for composing a (possibly relaxed) LCC C with a special type of PCP of proximity (PCPP). The result of the composition is a *relaxed* LCC C' which basically inherits the query complexity of the PCPP (and with a moderate overhead in block length).

Similarly to the relaxed LDCs of [7], the codewords of C' are constructed by taking repetitions of a codeword of a (possibly relaxed) robust[11] RLCC C and concatenating it with many PCPP proof strings. Specifically, for each set of queries that the relaxed local corrector for C would like to make, we write down a PCPP proof that this set of queries would make the corrector output the symbol that is in the position being corrected (had it made those queries). We shall refer to the first part of C', which contains repetitions of C, as the *core* of C', and refer to the second as the PCPP part.

Observe that the foregoing approach allows us to locally correct bits of the *core* of C'. The relaxed corrector for the composed RLCC takes the queries made by the old corrector as input, and uses the PCPP verifier to test if the old corrector would have accepted.[12] However, we shall need a more sophisticated machinery to correct the PCPP part of C' (indeed this is exactly the challenge that we faced when trying to follow the [7] approach). This will be achieved by ensuring that the PCPPs that we use have strong properties.

In particular, we shall employ the foregoing composition strategy while using the PCPPs of Section 1.1.1, which admit canonical proofs, strongly canonical soundness, and self-correctability. Recall that a PCPP is said to have strongly canonical soundness if every valid input has a *canonical* proof that it accepts, and any pair of statement and proof are rejected with probability proportional to their distance from a true statement and its corresponding canonical proof. In addition, recall that a canonical PCPP is said to be *self-correctable* if the canonical proof strings form a locally correctable code (i. e., if it is possible to locally recover individual bits of a noisy PCPP oracle).

Suppose that we want to correct a bit that lies in the PCPP part of a purported codeword of C'. If this bit is in a PCPP oracle that is not too corrupted, we can simply use the PCPP's self-corrector to recover the bit. However, as pointed out before, this naive attempt to self-correct fails when the *entire* proof string

[11] We define robust RLCCs formally in Section 4.1.2.

[12] Even for this to work, we need to ensure that the original RLCC is *robust*, in the sense that with high probability the corrector's view (i. e., the answers to its queries) are *far* from answers that would make it output an incorrect value. Otherwise, we have no guarantee that the PCPP verifier will see the error.

is corrupted. This can easily happen when the proof strings, each of which refers to a single possible query set of the original corrector, are short relative to the size of the entire codeword.

Thus, our main challenge is to detect whether the given PCPP proof string was (possibly entirely) corrupted. We observe that if on the one hand, the PCPP oracle we wish to correct is heavily corrupted, while on the other hand, the statement to which the PCPP refers (i. e., the queries that the corrector for C makes) is *not* heavily corrupted, then the proof is far from the prescribed canonical proof. The strongly canonical soundness guarantees that in such case the PCPP verifier will detect the corruption and reject. Thus, we are left with the case that *both* the PCPP oracle and the statement that it refers to are heavily corrupted.

To detect this deviation, we choose a random point in the foregoing statement and read it directly. Since that point is in the core of the code, and we have already described the procedure for correcting in the core, we can also correct this point and compare the corrected value with the symbol that we read directly. Since we have assumed that the statement was heavily corrupted, the value that we read directly will likely be different than what the corrector returns, in which case we can reject.

Equipped with this composition theorem, we can now construct our code. By applying the composition theorem to the low-degree extension code, of suitable parameters, and the PCPP given in Theorem 1.5, we can already construct a constant-query RLCC with quasi-polynomial block length. We note that this is already a considerable savings when compared to the current best (full-fledged) LCCs. However, to obtain polynomial block length, we shall perform two compositions with different PCPPs (in direct analogy to the first proof of the PCP theorem [2]).

As in the quasi-polynomial result mentioned above, our starting point is the low-degree extension code. Under a suitable parameterization, this code is known to be a robust (full-fledged) LCC with nearly linear block length and polylogarithmic query complexity. We shall first compose it with the polynomial length, polylogarithmic query, strongly canonical, self-correctable and robust PCPP of Theorem 1.6. Since the foregoing PCPP is robust, this composition yields a robust RLCC with polynomial block length and slightly sublogarithmic query complexity. Finally, we compose yet again with the exponential length, constant query, strongly canonical, self-correctable PCPP from Theorem 1.5, which yields an RLCC with constant query complexity.

Each of our two composition steps introduces at least a quadratic overhead to the block length. This is because our composition of an RLCC with a PCPP appends a PCPP proof-string for every pair (i,ρ) of coordinate i to be corrected from the base code and random string ρ of the underlying corrector with respect to the point i.[13] Since we apply two such composition steps, we get a code with roughly $n \approx k^4$ block length.

1.2.2 Constant-rate RLCC

For the constant-rate construction, we build on the recent breakthrough construction of locally testable[14] and correctable codes of Kopparty *et al.* [37]. Interestingly, we will actually focus on the [37] construction

[13] In contrast, in standard PCP composition, one only appends an inner PCP proof-string for every random string ρ of the outer PCP. Thus, as long as the randomness complexity of the outer PCP is minimal, it is possible to achieve close to constant multiplicative overhead when composing.

[14] Recall that a locally testable code [27] is a code for which one can test, using a sublinear number of queries, whether a given string is a codeword or far from such.

of locally *testable* codes (rather than correctable ones), even though our own goal is to construct (relaxed) locally *correctable* codes.[15]

Kopparty *et al.* construct locally testable codes with query complexity $\exp(O(\log\log n)^2)$ by taking an iterative approach, similar to that of Meir [40]. They start off with a code of dimension $\operatorname{poly}\log(n)$ (which is trivially locally testable, by reading the entire codeword) and gradually increase its block length, while (almost) preserving the local testability and maintaining the rate of the code close to 1. This amplification step is achieved by combining two transformations on codes:

1. *Code tensoring*: this transformation squares the block length (which is good since we want to obtain block length n) and rate (which is not too bad since our rate is close to 1). The main negative affect is that this transformation also squares the distance.

2. *Distance amplification*: remarkably, this transformation fixes the loss in distance caused by the tensoring step, without harming the rate or local testability too much.

As noted above, in their work, Kopparty *et al.* also construct a locally correctable code, albeit only with query complexity $\exp(\widetilde{O}(\sqrt{\log n}))$. The reason why their LCC construction does not match the parameters of their LTC construction is that the tensoring step, used in their construction of locally *testable* codes, is not known to preserve local correctability.[16]

Our key observation is that tensoring does preserve *relaxed* local correctability. Recall that the tensor of a code $C : \mathbb{F}^k \to \mathbb{F}^n$ is the code $C^2 : \mathbb{F}^{k^2} \to \mathbb{F}^{n^2}$ that consists of all strings $c \in \mathbb{F}^{n^2}$, viewed as $n \times n$ matrices, that consist of rows and columns that are each codewords of c.

Suppose that C is a (relaxed) LCC with query complexity q. We want to show that C^2 is also a (relaxed) LCC with query complexity roughly q. Let $w \in \mathbb{F}^{n^2}$ be a (possibly) corrupt codeword of C^2. Thus, w which we also view as an $n \times n$ matrix, is close to some codeword $c \in \mathbb{F}^{n^2}$. Given an index $(i, j) \in [n] \times [n]$ to correct, a natural approach is apply the (relaxed) local corrector of C on the i^{th} row of w, with respect to the index j.

If it were the case that the i^{th} row of w were close to the i^{th} row of c, we would be done. However, the i^{th} row of w only constitutes a $1/n$ fraction of w and so it could potentially be entirely corrupt. Let us assume that it is indeed the case that the i^{th} rows of c and w (almost) entirely disagree.

To detect that this is the case, our corrector chooses at random a few columns $J \subset [n]$. On the one hand, since the i^{th} rows of w and c disagree almost everywhere, with high probability for some $j' \in J$ it will be the case that $w_{i,j'} \neq c_{i,j'}$. On the other hand, since j' is just a random column, with high probability the j'^{th} columns of w and c are close.

Given this, a natural approach is to have our corrector robustly read the (i, j')-th entries of w for every $j' \in J$, by applying the (relaxed) local corrector of C on each column j'. In the likely case that it chooses a j' such that the j'^{th} column of w and c are close, with high probability the corrector will either return $c_{i,j'}$ or $\perp$. If our corrector sees $\perp$ it can immediately reject (since this would never happen for an exact codeword). Otherwise, if our corrector sees the value $c_{i,j'}$, it can compare this value with $w_{i,j'}$ (by

[15]This may not be surprising, since the notions of relaxed correctability and testability are closely related. In particular, as observed in [25], every RLDC (analogously, RLCC) is roughly equivalent to a code C such that for every coordinate i and value b, the subcode obtained by fixing the i'th bit to b (i. e., $\{C(x) : C(x)_i = b\}$) is locally testable.

[16]It can be shown that tensoring at most *squares* the query complexity for local correcting. However, the [37] iterative approach cannot afford such an overhead in each iteration.

explicitly reading the (i, j')'th entry of w). By the above analysis, these values will be different (with high probability), in which case our corrector can also reject.

To calculate the overall query complexity of the resulting code, we need to account for the overhead introduced by both the tensoring and distance amplification steps. Assuming that C is (relaxed) locally correctable up to distance δ_R, the tensoring step only increases the query complexity by $O(1/\delta_R)$. Each distance amplification increases the query complexity by roughly a $\text{polylog}(n)$ factor. Thus, since we need roughly $\log\log n$ iterations to reach block length n, the overall query complexity is $\exp(O(\log\log n)^2)$.

1.3 Related work

Subsequent to the initial publication of this work [30], the open problem raised in Section 1.1 was recently resolved by Chiesa, Gur, and Shinkar [12], who improved the block length in Theorem 1.1 from nearly quartic to nearly linear. More precisely, they showed that for every $\alpha > 0$, there exists an $O(1)$-query RLCC with block length $n = k^{1+\alpha}$, matching the parameters of state-of-the-art constructions of RLDC. Recent work [29, 13] shows lower bounds for RLCCs, which match the improved parameters of [12] nearly tightly.

A similar notion to RLDCs called *Locally Decode/Reject Codes (*LDRC*s)* arose in the beautiful work of Moshkovitz and Raz [42] on constructing two-query PCPs with subconstant error. These are similar to RLDCs in that they are codes with a decoder that is permitted to reject if it sees errors. However, it is important to note that the two notions differ in a few significant ways and are overall incomparable. First, LDRCs decode a k-tuple of coordinates jointly, rather than a single coordinate. Second, LDRCs have a “list-decoding” guarantee—namely, that the decoding agrees with one message in a short list of messages—as opposed to RLDCs, which provide unique decoding (but up to a smaller radius). Finally, LDRCs only need to work with high probability over the choice of k-tuple, while RLDCs must decode or reject with high probability for *every* coordinate. See [42, Section 2] for the formal definition of LDRCs and a comparison to RLDCs.

A closely related notion is that of *decodable* PCP*s* (dPCP), first introduced by Dinur and Harsha [15] to the end of obtaining a modular and simpler proof of the the [42] result. A dPCP is a PCP oracle, encoding an NP-witness, which allows for list decoding of individual bits of the NP witness it encodes. Dinur and Harsha provided constructions of such dPCPs as well as a composition theorem for dPCPs.

The notion of *locally recoverable codes* (LRCs) [28], which recently received much attention in the context of data storage (e. g., [49, 10, 6, 33]), is also closely related to that of locally correctable codes. Here, the goal is to correct *a single error* locally, obtaining dramatically better parameters than those obtainable via locally correctable codes, which can tolerate a linear number of errors.

Additionally, in a recent paper, Goldreich and Gur [25] introduced the notion of *universal locally testable codes* (universal-LTC), which can be thought of as generalizing the notion of relaxed LDCs. A universal-LTC $C : \{0,1\}^k \to \{0,1\}^n$ for a family of functions $\mathcal{F} = \left\{f_i : \{0,1\}^k \to \{0,1\}\right\}_{i\in[M]}$ is a code such that for every $i \in [M]$ and $b \in \{0,1\}$, membership in the subcode $\{C(x) \,:\, f_i(x) = b\}$ can be locally tested. As was shown in [25], universal-LTCs with respect to the family of dictator functions (i. e., functions of the form $f(x) = x_i$) are roughly equivalent to RLDCs. While universal-LTCs do not capture the notion of RLCC, one can consider a natural generalization of universal-LTCs in which the functions are defined on the codeword rather than message space, i. e., with respect to family of functions

$\mathcal{F} = \{f_i : \{0,1\}^n \to \{0,1\}\}_{i\in[M]}$. Such a generalization captures the notion of RLCC, however, it is not clear whether the results in [25] apply to this setting. We leave this question to future work.

Finally, we remark that the relaxed LDCs have been used in the context of interactive proofs of proximity [31] and property testing [11].

1.4 Organization

In Section 2 we provide standard definitions and notations. In Section 3 we formally define local correcting, and the relaxation that we introduce. In Section 4 we give our first construction, an RLCC with constant query complexity, while in Section 5 we give our second construction, an RLCC with constant rate.

2 Preliminaries

We denote the relative distance, over alphabet Σ, between two strings $x \in \Sigma^n$ and $y \in \Sigma^n$ by

$$\Delta(x,y) \stackrel{\text{def}}{=} \frac{|\{i \in [n] \,:\, x_i \neq y_i\}|}{n}.$$

If $\Delta(x,y) \leq \varepsilon$, we say that x is ε-close to y, and otherwise we say that x is ε-far from y. Similarly, we denote the relative distance of x from a non-empty set $S \subseteq \Sigma^n$ by $\Delta(x,S) \stackrel{\text{def}}{=} \min_{y\in S} \Delta(x,y)$. If $\Delta(x,S) \leq \varepsilon$, we say that x is ε-close to S, and otherwise we say that x is ε-far from S.

2.1 Error-correcting codes

Let $k < n$ be positive integers and let Γ, Σ be alphabets. A code $C : \Gamma^k \to \Sigma^n$ is an *injective* mapping from messages of length k (over the alphabet Γ) to codewords of length n (over the alphabet Σ). Typically it will be the case that $\Gamma = \Sigma$, in which case we simply say that the code is over the alphabet Σ. We refer to n as the block length of the code (which we think of as a function of k). The rate of the code is defined as k/n. The relative distance of the code is the minimum, over all pairs of distinct messages $x, y \in \Gamma^k$, of $\Delta(C(x), C(y))$. We shall sometimes slightly abuse notation and use C to denote the set of all of its codewords $\{C(x)\}_{x\in\Gamma^k} \subset \Sigma^n$.

Let $\mathbb{F}$ be a finite field (which we think of as an alphabet). We say that a code $C : \mathbb{F}^k \to \mathbb{F}^n$ is a linear code if it is a linear map from $\mathbb{F}^k$ to $\mathbb{F}^n$. In this case the set C of codewords is a subspace of $\mathbb{F}^n$.

2.1.1 Code concatenation

Code concatenation is an operation on codes that is commonly used to reduce alphabet size. Fix alphabets Σ, Ξ, and Γ. Fix an "outer" code $C : \Sigma^k \to \Xi^n$ with distance δ_C and rate r_C, and an "inner" code $D : \Xi \to \Gamma^r$ with distance δ_D and rate r_D. The concatenation of C with D is the code $C' : \Sigma^k \to \Gamma^{r\cdot n}$ such that each $x \in \Sigma^k$ is first encoded with C, and then each symbol of the resulting codeword is encoded using the code D. The relative distance of C' is at least $\delta_C \cdot \delta_D$ and the rate is $r_C \cdot r_D$.

Let $\mathbb{F}$ and $\mathbb{G}$ be finite fields, such that $\mathbb{G}$ is an extension field of $\mathbb{F}$. That is, $\mathbb{G} \cong \mathbb{F}^m$ for some $m \in \mathbb{N}$. Let $C : \mathbb{F}^k \to \mathbb{G}^n$ and $D : \mathbb{G} \to \mathbb{F}^r$ be error-correcting codes that are $\mathbb{F}$-linear (where we identify $\mathbb{G}$ with $\mathbb{F}^m$). Then the code obtained by concatenating C and D is $\mathbb{F}$-linear.

We will often concatenate our codes with good binary linear codes, that is, binary linear codes with constant rate and distance.

Lemma 2.1 ([36])**.** There exist (explicit) binary linear codes with constant rate and distance.

3 Definitions: local correcting and its relaxations

First, we define the notion of (full-fledged) local correctability for codes.

Definition 3.1 (Locally Correctable Codes (LCCs))**.** Let $C \subseteq \Sigma^n$ be an error correcting code with relative distance δ. We say that $\mathcal{C}$ is locally correctable if there exists a constant $\delta_{\mathsf{radius}} < \delta/2$, which we call the correcting radius, and a randomized polynomial time algorithm $\mathcal{M}$ that gets oracle access to a string $w \in \Sigma^n$ and explicit input $i \in [n]$. We require that if w is δ_{radius}-close to some codeword c, we have that $\mathcal{M}^w(i) = c_i$ with probability at least $2/3$. Furthermore, if w is a codeword, we require that $\mathcal{M}^w(i) = c_i$ with probability 1.[17]

The query complexity of $\mathcal{M}$ is the maximal number of queries that $\mathcal{M}$ makes for any input i and w.

(Note that the constant $2/3$ can be amplified as usual by repeating the process multiple times and outputting the majority symbol.)

Following Ben-Sasson *et al.* [7], in this article we consider a relaxation of LCCs, with the aim of constructing more efficient codes that use this relaxation. Loosely speaking, the relaxation allows the decoder to output a special abort symbol $\perp$ which indicates that it is unsure how to correct.

Definition 3.2 (Relaxed Locally Correctable Codes (RLCCs))**.** Let $C \subseteq \Sigma^n$ be an error correcting code. We say that $\mathcal{C}$ is relaxed locally correctable (RLCC) if there exists a constant $\delta_{\mathsf{radius}} \in (0,1]$, which we call the correcting radius, and a randomized polynomial time algorithm $\mathcal{M}$, which gets as input oracle access $w \in \Sigma^n$ and explicit access to an index $i \in [n]$, such that the following two conditions hold.

1. If $w \in \mathcal{C}$, then $\mathcal{M}^w(i) = w_i$ with probability 1.
2. If w is δ_{radius}-close to some codeword $c \in C$, then

$$\Pr\big[\mathcal{M}^w(i) \in \{c_i, \perp\}\big] \geq 1/2,$$

where $\perp \notin \Sigma$ is a special abort symbol.

Some remarks about Definition 3.2 are in order. First, we note that every LCC is, in particular, a relaxed LCC. Second, one might worry that Condition 2, which allows the corrector to state that it does not know how to decode, could allow the trivial decoder that simply responds with $\perp$ to everything. However, Condition 1 prevents this, by stipulating that the corrector must always succeed when given a valid codeword.

We conclude this section with two additional remarks.

[17] Requiring perfect correctness for exact codewords is non-standard, but it holds for state-of-the-art LCCs. We include this requirement here since it is required for RLCCs.

Remark 3.3 (RLCC Error Reduction)**.** Note that the probability of error in Definition 3.2 can be reduced by repeating the test t times (with independent coin tosses). If all the tests agree on a symbol $\sigma \neq \perp$ then the amplified corrector outputs σ and otherwise it outputs $\perp$. For the amplified corrector to make a mistake, all t iterations must fail, which happens with probability 2^{-t}.

Remark 3.4 (Success Rate)**.** Lastly, we mention a third condition that appears in the definition of relaxed local *decoders* in [7], which states that there is a constant fraction of coordinates $i \in [n]$ for which $\Pr[\mathcal{M}(w,i) = c_i] \geq 2/3$—namely, for which the relaxed local decoder successfully retrieves the symbol (and does not reject).

In the scenario for which our relaxed local corrector makes a *constant* number of queries, this property can be shown to follow directly from the two conditions in Definition 3.2, via a transformation that is analogous to the one in [7].

4 Constant-query RLCC

In this section we construct a relaxed locally correctable code with constant query complexity and polynomial length.

Theorem 4.1. *For every $\varepsilon > 0$, there exists a (linear) relaxed* LCC $C : \{0,1\}^k \to \{0,1\}^n$ *with constant relative distance, constant query complexity and block length $n = k^{4+\varepsilon}$.*

The key tool in the proof of Theorem 4.1 is a composition technique which shows that by combining a (relaxed) locally correctable code that is "robust" with a suitable PCP of proximity (PCPP), one obtains an RLCC with improved query complexity.[18] More specifically, we need a general purpose PCPP that has the following two properties:

1. **Canonical Soundness:** Each input $x \in \mathcal{L}$, where $\mathcal{L}$ is the target language, has a specific "canonical" PCPP proof string. Loosely speaking, the requirement is that even inputs that are in the language are rejected if the proof that is provided is not the canonical one. Canonical soundness was first defined by Goldreich and Sudan [27].

2. **Self Correction:** We require that the set of canonical PCPP proof strings forms an error correcting code (i. e., it has distance), and furthermore, that this code is locally correctable. In other words, symbols in the PCPP proof string can be reconstructed even if the proof string is slightly corrupted, using few queries.

Our composition is similar to (and inspired by) composition of PCPs, and especially by the approach advocated by Ben Sasson *et al.* [7] who compose any "robust" PCP with a PCP of proximity (PCPP).

For a high-level overview of our construction, see Section 1.2.1.

[18] Loosely speaking, a PCPP is similar to a PCP except that the verifier has oracle access both to the proof *and* to the main input. The soundness requirement is only that the verifier rejects inputs that are *far* from the language.

Section organization. In Section 4.1 we introduce several important definitions that will be used throughout this section, including the various strong notions of PCPPs that we use. In Section 4.2 we state and prove our composition theorem. We then proceed to construct the two PCPPs that we will use. In Section 4.3 we construct an “Hadamard-like” PCPP (which has exponential length but constant query complexity) and then in Section 4.4 we construct a “[5]-like” PCPP (with polynomial-length and polylogarithmic query complexity). Finally, in Section 4.5 we put it all together and prove Theorem 4.1.

4.1 Preliminaries

In this section we provide the required preliminaries for our construction. Specifically, we define strongly canonical PCPs of proximity with self-correctable oracles, and review basic properties of the low-degree extension code.

4.1.1 Canonical self-correctable PCPP

We begin by recalling the definition of PCPs of proximity, which loosely speaking, are PCPs in which the verifier is only allowed to make a small number of queries to *both* the statement and the proof, and soundness only means that, with high probability, the statement is close to a correct statement.

Definition 4.2 (Probabilistically Checkable Proof of Proximity (PCPP) [7, 16])**.** A probabilistically checkable proof of proximity (PCPP) for a set $S \subseteq \{0,1\}^n$ consists of a probabilistic verifier $\mathcal{V}$ that gets access to an input oracle $x \in \{0,1\}^n$ and to a proof oracle π. The verifier is required to satisfy the following properties:

- **Completeness:** For every $x \in S$, there exists a proof π such that when V is given access to the input oracle x and proof oracle π it accepts with probability 1.
- **Soundness:** For every $x \notin S$ and every proof string π, when V is given access to the input oracle x and proof oracle π it rejects with probability at least $\Omega(\Delta(x,S))$.

The query complexity q of the PCPP is the maximal number of queries that $\mathcal{V}$ makes, to both the input and proof oracles, on any input $x \in \{0,1\}^n$ and proof π. Similarly, the randomness complexity r of the PCPP is the maximal number of random bits that $\mathcal{V}$ uses given any input oracle $x \in \{0,1\}^n$ and proof oracle π.

In our constructions it will be important for us to use PCPPs that are *linear* in the following sense:

Definition 4.3 (Linear PCPP)**.** We say that a PCPP $(\mathcal{P},\mathcal{V})$ is linear if it satisfies the following two conditions:

1. The mapping $\mathcal{P}$ from inputs x to PCPP proof strings is a *linear* function.
2. The verifier $\mathcal{V}$’s checks amount to checking that the answers lie in some affine subspace.

We say that a PCPP has a *linear verifier* $\mathcal{V}$ if it satisfies condition 2 above: that is, deciding predicate D can be expressed as a conjunction of linear functions (i. e., a linear system).

We shall actually need a stronger notion of PCPPs that have *strong soundness with respect to a canonical proof*. Loosely speaking, strongly canonical PCPP are PCPPs with two additional requirements: (1) *canonicity*: for every true statement there exists a unique proof (called the *canonical proof*) that the verifier is required to accept, and any other proof (even for a correct statement) must be rejected, and (2) *strong soundness*: the PCPP verifier is required to be *proximity oblivious*, namely, it rejects any pair of statement and proof with probability that is related to its distance from a true statement and its corresponding canonical proof.

Definition 4.4 (Strongly Canonical PCPP [27])**.** A PCPP for a set S, with verifier $\mathcal{V}$, is said to be strongly canonical if there exists a (deterministic) function $\mathcal{P} : S \to \{0,1\}^*$ that maps each input $x \in S$ to a canonical proof $\pi = \mathcal{P}(x)$ such that

- **Completeness (with respect to $\mathcal{P}$):** For every $x \in S$, when $\mathcal{V}$ is given access to the input oracle x and proof oracle $\pi = \mathcal{P}(x)$ it accepts with probability 1.
- **Strong Soundness (with respect to canonical proofs):** For every $x \in \{0,1\}^n$ and every proof oracle π, when V is given access to the input oracle x and proof oracle π it rejects with probability $\Omega(\mu)$, where:
$$\mu \stackrel{\text{def}}{=} \min_{x' \in S} \left\{ \max \left(\Delta\left(x, x'\right), \Delta\left(\mathcal{P}(x'), \pi\right) \right) \right\}. \tag{4.1}$$

Before we proceed, we give some intuition about Equation (4.1). For simplicity, we will pretend here that the proof length is the same as the input length (i. e., $|\mathcal{P}(x)| = n$ for every $x \in S \cap \{0,1\}^n$). Recall that in a PCPP, the verifier makes few queries to the proof oracle *and* to the input oracle. Unfortunately, this means that our verifier will usually be unable to distinguish $x \notin S$ from some very close $x' \in S$. On the other hand, if on input $x \in S$, the verifier is presented with a purported proof π that is very close to the canonical proof $P(x)$, it cannot be expected to reject whp. If we think of the input x and proof π as being noisy versions of some "true" input x' and canonical proof $P(x')$, the probability that our PCPP verifier distinguishes (x, π) from $(x', P(x'))$ is

$$\Delta\left(x \circ \pi, x' \circ P(x')\right) = \Delta\left(x, x'\right) + \Delta\left(\mathcal{P}(x'), \pi\right) = \Theta\left(\max \left(\Delta\left(x, x'\right), \Delta\left(\mathcal{P}(x'), \pi\right) \right) \right). \tag{4.2}$$

Minimizing over all $x' \in S$ gives us Equation (4.1).[19]

Finally, we define (strongly canonical) PCPP oracles that are self-correctable as follows.

Definition 4.5 (Self-correctable PCPP)**.** A strongly canonical PCPP with prover strategy $\mathcal{P}$ and query complexity q is said to be self-correctable if the function $\mathcal{P}$ is a locally correctable code with at most q queries and constant correcting radius.[20]

[19] We use the RHS of Equation (4.2) rather than the LHS, since the RHS also handles the case that the length of the proof string differs from the length of the input.

[20] One could alternatively introduce an additional parameter that measures the number of queries that the PCPP's self-corrector does. However, since we already have a lot of parameters, we choose to bound the corrector's query complexity by the PCPP verifier's query complexity.

4.1.2 Robustness

Loosely speaking, a PCPP is said to have *robust soundness* [7] if instead of just asking that the PCPP verifier reject false statements (with high probability), we ask that the verifier's local view (i. e., the answers to all queries) be "far" from any local view that would have made it accept. We shall refer to such PCPPs as *robust* PCPPs.

For the following definition, it would be convenient to think of a PCPP verifier $\mathcal{V}$ as a procedure that chooses a random string ω, and generates: (1) an (ordered) sequence of q queries $I_\omega = (i_1,\ldots,i_q)$, and (2) a decision predicate $D_\omega : \{0,1\}^q \to \{0,1\}$. Given query access to input x and PCPP oracle π, the verifier queries $(x\circ\pi)|_{I_\omega}$ and outputs $D_\omega\big((x\circ\pi)|_I\big)$. We write $(I,D)\leftarrow\mathcal{V}$ to denote the queries and decision predicate (randomly) generated by $\mathcal{V}$ (and note that these do not depend on the input x nor the proof string π). We also write $(I_\omega,D_\omega)=\mathcal{V}(\omega)$ to denote the (fixed) query locations and decision predicate that $\mathcal{V}$ outputs given the random string ω. While slightly abusing notation, we also denote by D_ω the set of answers $\alpha\in\{0,1\}^q$ that satisfy D_ω.

Definition 4.6 (Robust Strongly Canonical PCPP)**.** For robustness parameter $\rho\in(0,1)$, a PCPP with canonical proof $\mathcal{P}$ and verifier $\mathcal{V}$ for a set S has ρ-*robust* strong soundness s (with respect to canonical proofs) if the following condition is satisfied:

1. ***Accepting Views are Far Apart:*** For *every* ω, the set D_ω of accepting views, is an error correcting code with distance ρ.

2. ***Robust Soundness:*** For every $x\in\{0,1\}^n$ and every proof oracle π, when $\mathcal{V}$ is given access to the input oracle x and proof oracle π it holds that
$$\Pr_{(I,D)\leftarrow\mathcal{V}}\Big[\Delta\big((x\circ\pi)|_I,D\big)>\rho\Big]\geq s\cdot\mu,$$
where:
$$\mu \stackrel{\text{def}}{=} \min_{x'\in S}\left\{\max\left(\Delta\left(x,x'\right),\Delta\left(\mathcal{P}(x'),\pi\right)\right)\right\}.$$

Intuitively, the strong soundness with respect to canonical proofs assures that the verifier $\mathcal{V}$ will reject statement-proof pairs with probability that is proportional to their distance from a valid pair (of statement $x'\in S$ and its canonical proof $\mathcal{P}(x')$), and the robustness additionally assures that the distance of $\mathcal{V}$'s local view from an accepting one is proportional to the aforementioned distance between the given statement-proof pair and a valid pair. We note that when the distance from $\mathcal{V}$'s local view to an accepting view is nonzero, the verifier always rejects, and so robust soundness is stronger than traditional soundness.

Next, we extend the foregoing notion of robustness to the setting of relaxed locally correctable codes. Note that relaxed LCCs admit correcting procedures rather than testing procedures, and so it is not immediately clear what soundness means in this context (let alone robust soundness). However, it is natural to define the robustness condition for RLCC as the requirement that with high probability the local view of the corrector is far from all local views that would have caused the (relaxed) corrector to output a wrong value.

We will think of a relaxed corrector as a procedure that, given an index $i\in[n]$, randomly generates an (ordered) sequence of q queries $I=(i_1,\ldots,i_q)$ and a function $D:\Sigma^q\to\Sigma\cup\{\bot\}$ (where $\bot\notin\Sigma$ is a special abort character) according to which the corrector decides; that is, the corrector's output with respect to a given word $w \in \Sigma^n$ to be corrected is $D(w|_I)$. We will also denote by $(I_{i,\omega}, D_{i,\omega}) = \mathcal{M}(i;\omega)$, the particular query set $I_{i,\omega}$ and decision function $D_{i,\omega}$ generated by $\mathcal{M}$ on input i and random string ω. We say that a RLCC has a linear corrector $\mathcal{M}$ if D is a procedure that checks that $w|_I$ satisfies a conjunction of linear constraints; if not $\mathcal{M}$ outputs $\perp$, and otherwise it outputs the evaluation of a linear function of $w|_I$. We proceed to define robust RLCC.

Definition 4.7 (Robust RLCC)**.** For robustness parameter $\rho \in (0,1)$ and soundness parameter $s \in (0,1)$, we say that an error correcting code LCC $C \subseteq \Sigma^n$, with correcting radius $\delta_{\mathsf{radius}} \in (0,1)$, is ρ-robust s-sound RLCC if there exists a correcting procedure $\mathcal{M}$, that on input $i \in [n]$ outputs a sequence of queries $I \in [n]^q$ and a decision predicate $D : \Sigma^q \to \Sigma \cup \{\perp\}$ such that the following conditions are satisfied:

1. ***Accepting Views are Far Apart:*** For every $i \in [n]$ and $\omega \in R$, the set $D^{-1}_{i,\omega}(\Sigma)$ is an error correcting code with distance ρ.

2. ***Robust Soundness:*** For every $i \in [n]$ and $w \in \Sigma^n$ that is δ_{radius}-close to a codeword $c \in C$, it holds that
$$\Pr_{(I,D) \leftarrow \mathcal{M}(i)}\left[\Delta\left(w|_I, D^{-1}\left(\Sigma \setminus \{c_i\}\right)\right) > \rho\right] \geq s,$$
where $I \in [n]^q$ is a set of query locations, $D : \Sigma^q \to \Sigma \cup \{\perp\}$ is the decision predicate and q is the query complexity.

Recall that $\perp \notin \Sigma$ and so the set $D^{-1}(\Sigma \setminus \{c_i\})$ above, refers to all answers that would have led the decoder to answer *incorrectly*. Intuitively, similarly to robustness of PCPs, the robust soundness condition in Definition 4.7 assures that with high probably the corrector will not only avoid outputting a wrong symbol, but also that its local view will be far from any local view that leads to outputting a wrong symbol.

We define *robust self-correctable* PCPP*s* similarly to Definition 4.5 except that we require that local correction procedure be robust.

4.1.3 Low-degree polynomials and the low-degree extension code

Let $\mathbb{F}$ be a finite field, $\mathbb{H} \subseteq \mathbb{F}$, and $m = m(k) \geq 1$ be a parameter, which we call the dimension. A basic algebraic fact is that for every function $f : \mathbb{H}^m \to \mathbb{F}$ there exists a *unique* function $\widetilde{f} : \mathbb{F}^m \to \mathbb{F}$ such that $\widetilde{f}$ is a polynomial with individual degree $|\mathbb{H}| - 1$ that agrees with f on $\mathbb{H}^m$. Moreover, there exists an individual degree $|\mathbb{H}| - 1$ polynomial $\beta : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ such that for every function $f : \mathbb{H}^m \to \mathbb{F}$ it holds that
$$\widetilde{f}(z) = \sum_{x \in \mathbb{H}^m} \beta(x,z) \cdot f(x).$$

The function $\widetilde{f}$ is called the low-degree extension of f (with respect to the field $\mathbb{F}$, subset $\mathbb{H}$ and dimension m).

The low-degree extension can also be viewed as an error-correcting code in the following way. Suppose that $\mathbb{H}$ and m are such that $|\mathbb{H}|^m = k$. Then, we can associate a string $x \in \mathbb{F}^k$ with a function $x : \mathbb{H}^m \to \mathbb{F}$ by identifying $\mathbb{H}^m$ with $[k]$ in some canonical way.

We define the low-degree extension of a string x as $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}(x) = \widetilde{x}$. That is, the function $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ is given as input the string $x \in \mathbb{F}^k$, views it as a function $x : \mathbb{H}^m \to \mathbb{F}$ and outputs its low-degree extension $\widetilde{x}$. The Polynomial Identity Lemma[21] (Lemma 4.8) implies that this code has relative distance $1 - \frac{m\cdot(|\mathbb{H}|-1)}{|\mathbb{F}|}$.

Lemma 4.8 (Polynomial Identity Lemma)**.** Let $P : \mathbb{F}^m \to \mathbb{F}$ be a *non-zero* polynomial of total degree d. Then,

$$\Pr_{r\in\mathbb{F}^m}\left[P(r) = 0\right] \le \frac{d}{|\mathbb{F}|}.$$

Self-correction of polynomials. We will use the fact that low-degree polynomials are self-correctable, with a corrector that is both linear and robust.

Lemma 4.9 (Self-Correction Procedure (see [19, 48]))**.** Let $\delta < 1/5$ and $d, m \in \mathbb{N}$ such that $d \le |\mathbb{F}|/10$. There exists an algorithm that, given $x \in \mathbb{F}^m$ and oracle access to an m-variate function $P : \mathbb{F}^m \to \mathbb{F}$ that is δ-close to a polynomial P' of total degree d, makes $O(d)$ oracle queries and outputs $P'(x)$ with probability $2/3$. Moreover, if P has total degree d, then given $x \in \mathbb{F}^m$, the algorithm outputs $P(x)$ with probability 1.

Furthermore, with probability $2/3$, the answers that the algorithm receives to its queries are 5δ far from values that would lead it to output any value other than $P(x)$, and the algorithm computes a linear function of the symbols it queries.

The self-correction algorithm samples a random line ℓ_x in $\mathbb{F}^m$ passing through the point x, and runs the Berlekamp-Welch corrector for univariate polynomials of degree at most d on ℓ_x. To get robustness, we additionally read $O(d)$ extra points along the line (more than the Berlekamp-Welch corrector requires), and check that they match the decoded polynomial. Since each point read is individually random, the probability that we read a set of points that is 5δ-close to ones that is consistent with a different degree-d polynomial can be made arbitrarily small via Markov's inequality. To establish that the test is linear, observe that the Berlekamp-Welch algorithm is a linear function of the symbols read.

Low-degree tests. The celebrated low-degree test is a key operation in most PCP constructions. We will need several variants of this test which are listed below. The most standard one is the following:

Lemma 4.10 (Total Degree Test (a.k.a. Low Degree Test) (see, e. g., [46, 18]))**.** Let $\varepsilon \in (0, 1/2)$ and $d, m \in \mathbb{N}$ such that $d \le |\mathbb{F}|/2$. There exists an algorithm that, given oracle access to an m-variate function $P : \mathbb{F}^m \to \mathbb{F}$, makes $O(d \cdot \mathrm{poly}(1/\varepsilon))$ queries and:

1. Accepts every function that is a polynomial of *total* degree d with probability 1; and
2. Rejects functions that are ε-far from every polynomial of total degree d with probability at least $1/2$.

We will also need a more refined version of the test that tests the individual degree of the polynomial.

[21] This lemma, best known under the name "Schwartz–Zippel Lemma," goes back to a 1922 paper by Ore and has been rediscovered several times since. This history was uncovered by Arvind et al. [4]. The name "Polynomial Identity Lemma" was coined by Babai and has been adopted by [4] and two other papers.

Lemma 4.11 (Individual Degree Test (see, e. g., [31, Theorem A.8])**.** Let $d,m \in \mathbb{N}$ such that $dm < |\mathbb{F}|/10$ and $\varepsilon \in (0,1/10)$. There exists an algorithm that, given oracle access to an m-variate polynomial $P : \mathbb{F}^m \to \mathbb{F}$, makes $O(dm \cdot \mathrm{poly}(1/\varepsilon))$ queries, and:

1. Accepts every function that is a polynomial of *individual* degree d with probability 1; and
2. Rejects functions that are ε-far from every polynomial of individual degree d with probability at least $1/2$.

Lastly, we require a robust analog of the low-degree test.

Lemma 4.12 (Robust Low Degree Test [18])**.** Let $d,m \in \mathbb{N}$ such that $d \leq |\mathbb{F}|/100$. Let $\ell : \mathbb{F} \to \mathbb{F}^m$ be a random line. Then, for any m-variate function $P : \mathbb{F}^m \to \mathbb{F}$ that is δ-far from having degree d it holds that

$$\mathbf{E}_\ell\big[\Delta(P \circ \ell, \text{degree } d \text{ univariate polynomials})\big] \geq \Omega(\delta),$$

where $P \circ \ell$ denotes the univariate polynomial obtained by composing P with ℓ.

4.2 Composition of relaxed LCC and PCPPs

In this section we prove a composition theorem for robust RLCCs with self-correctable PCPPs, which loosely speaking, yields a robust RLCC with improved query complexity, at a relatively small increase to the block length. We recall that composition consists of appending PCPP proofs to the RLCC, one for each set of queries that the corrector for the code would make. We now state the parameters of our composition formally.

Theorem 4.13 (Composition of Robust RLCC with (suitable) PCPP)**.** *Let $\mathbb{F}$ be a finite field, and let $C : \mathbb{F}^k \to \mathbb{F}^n$ be a linear ρ_{code}-*robust *relaxed* LCC*, with respect to soundness s_{code}, with a linear corrector, relative distance δ, correcting radius $\delta_{\mathsf{radius}} < \delta/2$, randomness complexity r_{code}, and query complexity q_{code}.*

*Let $(\mathcal{P},\mathcal{V})$ be a linear ρ_{pcp}-*robust *strongly canonical* PCPP *for inputs of length $2q_{\mathsf{code}}$, with respect to soundness s_{pcp}, for affine relations, with a linear verifier, randomness complexity r_{pcp}, query complexity q_{pcp}, self-correction radius $\delta_{\mathsf{pcp\text{-}correction\text{-}radius}} \geq \rho_{\mathsf{pcp}}$, self-correction robustness $\rho_{\mathsf{self\text{-}correct}} \geq \rho_{\mathsf{pcp}}$, self-correction soundness $s_{\mathsf{self\text{-}correct}} \geq s_{\mathsf{pcp}}$, and self-correction query complexity $q_{\mathsf{self\text{-}correct}} = q_{\mathsf{pcp}}$.*

The composition of the outer code C with the inner PCPP $(\mathcal{P},\mathcal{V})$ *yields a linear* $(\rho_{\mathsf{pcp}}/5)$-robust *relaxed* LCC $C' : \mathbb{F}^k \to \mathbb{F}^{n'}$*, with respect to soundness $s' = (s_{\mathsf{pcp}} \cdot s_{\mathsf{code}} \cdot {\rho_{\mathsf{code}}}^2)/8$, with a linear corrector, block length $n' = 2n \cdot 2^{r_{\mathsf{code}}} \cdot 2^{r_{\mathsf{pcp}}} \cdot q_{\mathsf{pcp}}$, relative distance $\delta/2$, correcting radius $\delta_{\mathsf{radius}}/4$, randomness complexity $2r_{\mathsf{code}} + O\big(r_{\mathsf{pcp}} + \log(q_{\mathsf{pcp}}) + \log(q_{\mathsf{code}})\big)$, and query complexity $5q_{\mathsf{pcp}}$.*

In the following subsections we prove Theorem 4.13. Specifically, In Section 4.2.1 we describe the construction of the composed RLCC. In Section 4.2.2 we describe the relaxed corrector. In Section 4.2.3 we establish the basic properties of the code (distance, length, linearity, query complexity, and robustness). Then, in Sections 4.2.4 and 4.2.5 we establish the self-correctability of the "code part" and "PCPP" part of the composed code.

4.2.1 The construction of the composed code

Let $C : \mathbb{F}^k \to \mathbb{F}^n$ be the RLCC stated in Theorem 4.13. Let $\mathcal{M}$ be the relaxed correcting procedure associated with C. Let $R = \{0,1\}^{r_{\mathsf{code}}}$, and for every index $i \in [n]$ and $\omega \in R$, denote by $I_{i,\omega}$ (resp., $D_{i,\omega}$) the query set (resp., decision predicate), respectively, generated by $\mathcal{M}$ on input the index i and the random string ω. (Recall that this means that given oracle access to a purported codeword w, location $i \in [n]$, and randomness $\omega \in R$, the corrector $\mathcal{M}$ outputs $D_{i,\omega}(w|_{I_{i,\omega}})$).

For every affine subspace $S \subseteq \{0,1\}^n$, let $(\mathcal{P}_S, \mathcal{V}_S)$ be the PCPP from the theorem's statement. Let $\mathcal{M}_{\mathsf{PCP}}$ denote the self-correction procedure associated with the PCPP (where we omit the dependence of the corrector on S from the notation.)

The composed code $C' : \mathbb{F}^k \to \mathbb{F}^{n'}$ (for a block length n' as in the theorem's statement) is constructed as follows. Given an input $x \in \{0,1\}^k$, it's encoding $C'(x)$ consists of two parts:

1. **The code's core:** The core part consists of t copies of $C(x)$, where t will be set below such that the core and the PCPP part have the same length.[22]

2. **The code's PCPP:** The second part of the code will consist of a sequence of PCPP proof strings. For every $i \in [n]$ and every $\omega \in R$, let $(I_{i,\omega}, D_{i,\omega}) = \mathcal{M}(i;\omega)$. We define a set $S_{i,\omega}$ as follows:

$$S_{i,\omega} = \left\{ (z_1, z_2) \in (\mathbb{F}^{q_{\mathsf{code}}})^2 \ : \ z_1 = \sigma^{q_{\mathsf{code}}} \text{ for some } \sigma \in \mathbb{F}, \text{ and } D_{i,\omega}(z_2) = \sigma \right\}.$$

   or in words, the set $S_{i,\omega}$ consists of all pairs $(z_1, z_2) \in (\mathbb{F}^{q_{\mathsf{code}}})^2$ such that (1) z_1 consists of q_{code} copies of some element $\sigma \in \mathbb{F}$, and (2) $z_2 \in \mathbb{F}^{q_{\mathsf{code}}}$ makes $\mathcal{M}(i;\omega)$ output the symbol σ given answers z_2.

   Observe that since $\mathcal{M}$ is $\mathbb{F}$-linear, the set $S_{i,\omega}$ is an affine subspace. Hence, by the theorem's hypothesis it has a strongly canonical and locally-correctable PCPP $(\mathcal{P}_{i,\omega}, \mathcal{V}_{i,\omega})$. We let $\pi_{i,\omega} = \mathcal{P}_{i,\omega}\Big(\big((C(x)[i])^{q_{\mathsf{code}}}, C(x)|_{I_{i,\omega}} \big) \Big)$, for each $i \in [n]$ and $\omega \in R$.

Putting both parts together, we define $C' : \mathbb{F}^k \to \mathbb{F}^{n'}$ as follows

$$C'(x) = \Big((C(x))^t, (\pi_{i,\omega})_{i \in [n], \omega \in R} \Big),$$

where t is chosen such that $|C(x)|^t = \sum_{i \in [n], \omega \in R} |\pi_{i,\omega}|$ (i. e., the core and the PCPP parts have the same length).

4.2.2 The relaxed corrector

Consider a string $w \in \mathbb{F}^{n'}$ (which we think of as a noisy codeword for the correcting procedure). We view w as a string composed of two parts (analogous to the two parts of the construction above):

1. $(w_1, \ldots, w_t) \in (\mathbb{F}^n)^t$: the t alleged repetitions of some codeword in C (i. e., the code's core).

[22] As usual, integrality issues can be resolved via padding.

2. $\bar{\pi} = (\pi_{i,j})_{i\in[n],j\in[R]}$: the alleged canonical PCPP oracles asserting each one of the affine relations $(A_{i,j}, b_{i,j})_{i\in[n],j\in[R]}$.

We construct a relaxed local corrector $\mathcal{M}'$ that given a location $\ell^* \in [n']$ and oracle access to the purported codeword string $w = (\bar{c}, \bar{p})$, either corrects at the point ℓ^* or rejects. (Here and below, we use the convention that starred indices refer to locations that are non-random, whereas non-starred indices are random.) Our correctors works differently, depending on whether ℓ^* belongs to the core or to the PCPP part:

- **Correcting in the Core (i. e., $\ell^* \in [n \cdot t]$):**
  1. Let $i^* \in [n]$ be $((\ell^* - 1) \mod n) + 1$, where we recall that $n = |C(x)|$ is the original block length. In other words, i^* is the internal index within the (alleged) codeword of C that ℓ^* refers to.
  2. Select at random $j \in [t]$. Read the value $w_j[i^*]$ q_{pcp} times and check that the value read is always the same.[23]
  3. Uniformly draw $\omega \in R$, corresponding to the randomness of C's corrector $\mathcal{M}$. Let $I_{i^*,\omega}$, and $D_{i^*,\omega}$ be the query set and decision predicate, respectively, of $\mathcal{M}(i^*, \omega)$.
  4. Run the PCPP verifier $\mathcal{V}_{i^*,\omega}$ that was defined above, with respect to the input oracle (z_1, z_2), where $z_1 = (w_j[i^*])^{q_{\mathsf{code}}}$ and $z_2 = w_j|_{I_{i^*,\omega}}$, and the PCPP oracle $\pi_{i^*,\omega}$. If the PCPP verifier rejects then output $\perp$.
  5. Output $w_j[i^*]$.

- **Correcting the PCPP Part (i. e., $\ell^* \in [n \cdot t + 1, n']$):**
  1. Let π_{i^*,ω^*} be the PCPP oracle that ℓ^* refers to. Let I_{i^*,ω^*} and D_{i^*,ω^*} be the query set and decision predicate, respectively, of $\mathcal{M}$ on index i^* and random coins ω^*.
  2. Invoke the PCPP verifier $\mathcal{V}_{i^*,\omega^*}$ with respect to input oracle (z_1, z_2), where $z_1 = (w_1[i^*])^{q_{\mathsf{code}}}$ and $z_2 = w_1|_{I_{i^*,\omega^*}}$, and the PCPP oracle π_{i^*,ω^*}, and output $\perp$ if it rejects.[24]
  3. Choose a random location $i' \in I_{i^*,\omega^*}$. Read the value $w_1[i']$ q_{pcp} times and check that the value read is always the same. Denote this value by η.
  4. Invoke the relaxed corrector (recursively) with respect to location i' (note that this location is in the core, which was handled above), and output $\perp$ if it does not output η.
  5. Invoke the PCP's self-corrector $\mathcal{M}_{\mathsf{PCP}}$ on the location that corresponds to ℓ^* in π_{i^*,ω^*}, and output whatever it returns.

[23]This step, which at first may seem silly, is meant to ensure *robust* soundness. Indeed, when analyzing robustness, we must consider an adversary that is allowed to modify answers to certain queries in retrospect.

[24]The choice of w_1 here (rather than any other w_j) was arbitrary.

4.2.3 Basic properties of the composed code

In this subsection we analyze the basic properties of the composed code C' and its corrector, as defined in Section 4.2.1, and show they satisfy all the conditions of Theorem 4.13, except for the (robust) soundness of the relaxed corrector with respect to correcting radius $\delta'_{\mathsf{radius}}$, which will be established in the subsequent subsections.

Block length. The code $C'(x)$ consists of two parts: (1) the codeword $C(x)$ repeated t times, which makes it equal in length to the second part, and (2) a PCPP for every $i \in [n]$ and $\omega \in R$, where each PCPP refers to a statement of length $2q_{\mathsf{code}}$, and where t is selected such that both parts (1) and (2) are of equal length. Recall that $R = 2^{r_{\mathsf{code}}}$ and so the length of each PCPP oracle with respect to statements of length $2q_{\mathsf{code}}$ is $2^{r_{\mathsf{pcp}}} \cdot q_{\mathsf{pcp}}$. Hence the block length of C' is $n' = 2n \cdot 2^{r_{\mathsf{code}}} \cdot 2^{r_{\mathsf{pcp}}} \cdot q_{\mathsf{pcp}}$.

Randomness complexity. For correcting in the core, $\log(t)$ random bits are used to select j, where

$$t = \frac{n'}{2n} = \frac{2n \cdot 2^{r_{\mathsf{code}}} \cdot 2^{r_{\mathsf{pcp}}} \cdot q_{\mathsf{pcp}}}{2n} = 2^{r_{\mathsf{code}}} \cdot 2^{r_{\mathsf{pcp}}} \cdot q_{\mathsf{pcp}} \,.$$

Then, r_{code} random bits are needed to uniformly choose $\omega \in R$, and r_{pcp} random bits are used by the PCPP verifier. This totals in $2r_{\mathsf{code}} + 2r_{\mathsf{pcp}} + \log(q_{\mathsf{pcp}})$ randomness complexity.

For correcting the PCPP part, by the above analysis $2r_{\mathsf{code}} + 2r_{\mathsf{pcp}} + \log(q_{\mathsf{pcp}})$ random bits are needed for the recursive invocations of the corrector with respect to the core. In addition, at most $\log(q_{\mathsf{code}})$ random bits are needed to uniformly choose the location $i' \in I_{i^*,\omega^*}$, and r_{pcp} for the invocation of the PCPP verifier. Lastly the PCPP self-corrector also uses r_{pcp} randomness.

Hence the total randomness complexity is upper bounded by

$$2r_{\mathsf{code}} + O\big(r_{\mathsf{pcp}} + \log(q_{\mathsf{pcp}}) + \log(q_{\mathsf{code}})\big).$$

Query complexity. For correcting a point in the core, we perform q_{pcp} repeated queries to read $w_j[i^*]$. Then, q_{pcp} queries for invoking the PCPP on a statement of length $2q_{\mathsf{code}}$ (the corrector's queries).

For correcting a point in the PCPP part of C', we (recursively) invoke the corrector with respect to the core, which, by the above analysis, takes $2q_{\mathsf{pcp}}$ queries. In addition, we invoke the PCPP verifier and self-correction procedure for a total of $2q_{\mathsf{pcp}}$ additional queries. We also read $w_1[i']$ q_{pcp} times. Overall, the total query complexity is $5q_{\mathsf{pcp}}$.

Distance. Half of each codeword of C' consists of repetitions of the code C, which has relative distance δ. Hence, the the relative distance of C' is at least $\delta' \geq \delta/2$.

Linearity. The linearity of the code C' is immediate from the construction: The first part of each codeword of C' consists of repetitions of a codeword of C, which is a *linear* code, and the second part consists of *linear* PCPP oracles that refer to the first part (i. e., each PCPP oracle is a linear encoding of the first part). The linearity of the corrector of C' also follows by construction, since the predicate C' checks is the one induced by the invocation of the *linear* PCPP verifier $M_{\mathcal{P}}$ to verify the linear claims of the *linear* corrector of C.

4.2.4 Correcting the code's core

In this subsection we analyze the relaxed corrector presented in Section 4.2.2 in the case that the location ℓ^* (to be corrected) is in the code's *core* (i. e., $\ell^* \in [n \cdot t]$).

The first condition of relaxed correctors (successfully correct uncorrupted codewords with probability 1) follows easily from the construction of the corrector and the completeness of the underlying PCPP. To elaborate, since the codeword is completely uncorrupted, the string $z_2 = w_j|_{I_{i^*,\omega^*}}$ leads the corrector to output $w_j[i^*]$, due to the completeness of C, the "core" RLCC. Therefore, the PCPP verifier will always accept in this case and we will always output $w_j[i^*]$. We proceed to demonstrate that the correcting procedure $\mathcal{M}'$ additionally satisfies the properties of a robust RLCC (Definition 4.7). First, we will show that accepting views for $\mathcal{M}'$ are far apart, which will follow from the robustness of the underlying PCPP.

Lemma 4.14 (Accepting Views are Far Apart (in the core part))**.** For every $i \in [n \cdot t]$ and random string ω' for $\mathcal{M}'$, the accepting views of $\mathcal{M}'$ have distance $\rho_{\mathsf{pcp}}/2$.

Proof. Fix a random string ω' for $\mathcal{M}'$ and an accepting view $a \in \mathbb{F}^{2q_{\mathsf{pcp}}}$. Recall that the view of $\mathcal{M}'$ (when querying the core) consists of two parts $(a_1, a_2) \in \mathbb{F}^{2q_{\mathsf{pcp}}}$. The corrector $\mathcal{M}'$ checks that $a_1 = \sigma^{q_{\mathsf{pcp}}}$. Hence, to change the first part of a_1 to some different a_1', one must change *all* of it. As for a_2, by the fact that the PCPP is robust, one must change at least a ρ_{pcp} fraction of get an accepting a_2'. □

To complete the proof (of robustness for the core part) we still need to show robust soundness. This is proved in the following lemma.

Lemma 4.15 (Robust Soundness (in the core part))**.** Let $w' \in \mathbb{F}^{n'}$ be $(\delta_{\mathsf{radius}}/4)$-close to some codeword $c' \in C'$ and let $\ell^* \in [n \cdot t]$. Then,

$$\Pr_{(I,D) \leftarrow \mathcal{M}'(\ell^*)}\left[\Delta\left(w'|_I, D^{-1}\left(\mathbb{F} \setminus \{c'[\ell^*]\}\right)\right) > \rho_{\mathsf{pcp}}/2\right] \geq s_{\mathsf{code}} \cdot s_{\mathsf{pcp}} \cdot \rho_{\mathsf{code}}/4.$$

Proof. Since $c' \in C'$, there exists some $x \in \mathbb{F}^k$ such that the core of c' consists of t copies of $C(x)$.

Recall that we used $(w_1, \dots, w_t)$ to denote the core of w'. We first argue that with constant probability (over the choice of $j \in [t]$), it holds that w_j is δ_{radius}-close to $C(x)$.

Claim 4.16.

$$\Pr_{j \in [t]}\left[\Delta(w_j, C(x)) \leq \delta_{\mathsf{radius}}\right] \geq \frac{1}{2}.$$

Proof. Since the core $(w_1, \dots, w_t)$ is half of the length of w', it holds that $\Delta((w_1, \dots, w_t), (C(x))^t) \leq 2\Delta(w', c') \leq \delta_{\mathsf{radius}}/2$. By Markov, this implies that for at most half of the indices $j \in [t]$ it holds that $\Delta(w_j, C(x)) > \delta_{\mathsf{radius}}$. □

Therefore, throughout the rest of the proof we can fix j such that w_j is δ_{radius}-close to $C(x)$ (and this only costs us at most a constant factor in the success probability of the corrector). Note that since it has correcting radius δ_{radius}, the relaxed local corrector is guaranteed to work for such w_j.

Let i^* be the internal index associated with ℓ^*; that is, that $w'[\ell^*]$ refers to the same symbol as $w_j[i^*]$. We first consider the case that the location i^* that we are trying to correct is not "corrupted" in w_j.

Claim 4.17. *If $w_j[i^*] = C(x)[i^*]$, then the view of $\mathcal{M}'$ is $1/2$-far from any view that would make it output an incorrect answer (i. e., an answer in $\mathbb{F} \setminus \{w_j[i^*]\}$).*

Proof. First observe that by construction, the corrector $\mathcal{M}'$, always outputs either $w_j[i^*]$ or $\perp$. Since exactly half of $\mathcal{M}$'s queries were to $w_j[i^*]$, all of these queries would need to be modified to make it answer incorrectly. □

Thus, we may assume without loss of generality that $w_j[i^*] \neq C(x)[i^*]$. In this case, all the queries to $w_j[i^*]$ are consistent with making it output an incorrect answer (i. e.not $C(x)[i^*]$). Our goal now will be to show that, in this case, the answers to the queries to our robust PCPP will be far from the set of answers that would make it accept.

We now want to argue that the input on which we run the PCPP is far from an accepting input.

Claim 4.18. *$w_j|_{I_{i^*,\omega}}$ is ρ_{code}-far from $D_{i^*,\omega}^{-1}\left(\mathbb{F} \setminus \{C(x)[i^*]\}\right)$, with probability at least s_{code}.*

Proof. Recall that C is a ρ_{code}-robust relaxed LCC with soundness s_{code}. Since $\Delta(w_j, C(x)) \leq \delta_{\mathsf{radius}}$, by definition we have the following:

$$\Pr_{\omega \in R}\left[\Delta\left(w_j|_{I_{i^*,\omega}}, D_{i^*,\omega}^{-1}\left(\mathbb{F} \setminus \{C(x)[i^*]\}\right)\right) > \rho_{\mathsf{code}}\right] \geq s_{\mathsf{code}},$$

where $(I_{i^*,\omega}, D_{i^*,\omega}) = \mathcal{M}(i;\omega)$. □

Using the fact that the input to the PCPP is far from accepting (as shown in Theorem 4.18) and that the PCPP has robust soundness, we show that the view of $\mathcal{M}'$ is far from any that would not make it reject.

Claim 4.19. *With probability $s_{\mathsf{pcp}} \cdot s_{\mathsf{code}} \cdot \rho_{\mathsf{code}}/2$ the view of $\mathcal{M}'$ is $\rho_{\mathsf{pcp}}/2$-far from any view that would not make it reject.*

Proof. Recall that we run the PCPP verifier $\mathcal{V}_{i^*,\omega}$ with respect to the input oracle (z_1, z_2), where $z_1 = (w_j[i^*])^{q_{\mathsf{code}}}$ and $z_2 = w_j|_{I_{i^*,\omega}}$, and PCPP proof string $\pi_{i^*,\omega}$.

The robust soundness of our PCP implies that with probability at least $s_{\mathsf{pcp}} \cdot \mu$, the local view of the PCPP verifier will be ρ_{pcp}-far from any view that would make it accept, where

$$\mu \stackrel{\text{def}}{=} \min_{(z_1',z_2') \in S_{i^*,\omega}} \left\{ \max\left(\Delta\left((z_1,z_2),(z_1',z_2')\right), \Delta\left(\mathcal{P}(z_1',z_2'), \pi_{i^*,\omega}\right) \right) \right\}.$$

Let $(z_1', z_2') \in S_{i^*,\omega}$ that minimize μ. Notice that $z_1' = \sigma^{q_{\mathsf{code}}}$, for some $\sigma \in \mathbb{F}$. If $\sigma \neq w_j[i^*]$, then $\mu \geq \Delta((z_1,z_2),(z_1',z_2')) \geq \frac{1}{2}$ (since the entire first half needs to be changed). Thus, we may assume that $\sigma = w_j[i^*]$.

The fact that $(z_1', z_2') \in S_{i^*,\omega}$ means that $z_2' \in D_{i^*,\omega}^{-1}(\mathbb{F} \setminus \{C(x)[i^*]\})$. Thus, by Theorem 4.18, with probability s_{code}, it holds that $w_j|_{I_{i^*,\omega}}$ is ρ_{code}-far from z_2'. Assuming that the latter happens we have that $\mu \geq \Delta((z_1,z_2),(z_1',z_2')) \geq \rho_{\mathsf{code}}/2$. We conclude that in any case $\mu \geq \rho_{\mathsf{code}}/2$.

The claim follows by observing that the PCPP queries that $\mathcal{M}$' makes account for half of its total queries. □

This concludes the proof of Lemma 4.15. □

4.2.5 Correcting the PCPP Part

In this subsection we complete the analysis of the relaxed corrector presented in Section 4.2.1, by analyzing it in the case that the location (to be corrected) is in the code's PCPP part.

As in the previous case, the first condition of relaxed correctors is immediate from the construction. Specifically, the fact that C is an RLCC tells us that $D_{i^*,\omega^*}(z_2) = w_1[i^*]$, and so it follows from the perfect completeness of the PCPP that we will never reject in Step 2. The completeness of the relaxed corrector for the "core" that was established in the previous section, which guarantees we will never reject in Step 4. Finally, the PCP's self-corrector will always output the correct value on an uncorrupted codeword, so we will not output the wrong thing in Step 5.

As before, we first show that accepting views are far apart.

Lemma 4.20 (Accepting Views are Far Apart (in the PCPP part))**.** For every $i \in [n \cdot t + 1, n']$ and random string ω' for $\mathcal{M}'$, the accepting views of $\mathcal{M}'$ have distance $\rho_{\mathsf{pcp}}/5$.

Proof. Fix a random string ω' for $\mathcal{M}'$ and an accepting view $a \in \mathbb{F}^{5q_{\mathsf{pcp}}}$. Recall that the view of $\mathcal{M}'$ (when querying the PCPP) consists of four parts $(a_1, a_2, a_3, a_4) \in \mathbb{F}^{q_{\mathsf{pcp}}} \times \mathbb{F}^{q_{\mathsf{pcp}}} \times \mathbb{F}^{2q_{\mathsf{pcp}}} \times \mathbb{F}^{q_{\mathsf{pcp}}}$, where: (1) a_1 corresponds to the PCPP answers (wrt the invocation of $\mathcal{V}_{i^*,\omega^*}$), (2) a_2 corresponds to reading $w_1[i^*]$ q_{pcp} times (and checking that all queries were the same), (3) a_3 corresponds to the answers for the recursive invocation of the corrector on the core, and (4) a_4 corresponds to the answers for the PCPP self-correction procedure.

We observe that each of these parts is robust by itself, where the first part is ρ_{pcp}-robust since the PCPP is ρ_{pcp}-robust, the second part is 1-robust (since you need to change all answers to ensure consistency), the third part is $\rho_{\mathsf{pcp}}/2$-robust by Lemma 4.15 and the fourth part is ρ_{pcp}-robust by the robustness of the PCPP self-correction.

Overall we get that accepting views are at least $(\rho_{\mathsf{pcp}}/5)$-far from each other. □

In the rest of this subsection we prove the following lemma, which shows that the second condition (robust soundness) is satisfied.

Lemma 4.21. For every string $w' = \left((w_1, \ldots, w_t), (\pi_{i,\omega})_{i \in [n], \omega \in R}\right)$ that is $(\delta_{\mathsf{radius}}/4)$-close to a valid codeword $c' \in C'$, and index $\ell^* \in [n \cdot t + 1, \ldots, n']$, the following inequality holds:

$$\Pr_{(I,D) \leftarrow \mathcal{M}'(\ell^*)}\left[\Delta\left(w'|_I, D^{-1}(\mathbb{F} \setminus \{c'[\ell^*]\})\right) > \rho_{\mathsf{code}}/5\right] \geq (s_{\mathsf{code}} \cdot s_{\mathsf{pcp}} \cdot {\rho_{\mathsf{code}}}^2)/8.$$

Proof. Let $i^* \in [n]$ and $\omega^* \in R$ be the indices that are associated with ℓ^* (that is, such that π_{i^*,ω^*} is the PCPP oracle in which the index ℓ^* resides). Let $x \in \mathbb{F}^k$ such that $C'(x) = c'$. Recall that the core of c' consists of t copies of $C(x)$.

Let Good denote the event that the view of $\mathcal{M}'$ is $(\rho_{\mathsf{pcp}}/5)$-far from any view that would make it return an incorrect value (i. e., different from $C'(x)[\ell^*]$ or $\perp$).

We will show that $\Pr[\mathsf{Good}] \geq (s_{\mathsf{code}} \cdot s_{\mathsf{pcp}} \cdot {\rho_{\mathsf{code}}}^2)/8$. This is done by a careful (and somewhat tedious) case analysis.

Claim 4.22. *If $w_1|_{I_{i^*,\omega^*}}$ is $(\rho_{\mathsf{code}}/2)$-far from $C(x)|_{I_{i^*,\omega^*}}$, then* $\Pr[\mathsf{Good}] \geq (s_{\mathsf{code}} \cdot s_{\mathsf{pcp}} \cdot {\rho_{\mathsf{code}}}^2)/8$.

Proof. If $w_1|_{I_{i^*,\omega^*}}$ is $(\rho_{\mathsf{code}}/2)$-far from $C(x)|_{I_{i^*,\omega^*}}$ then, with probability $\rho_{\mathsf{code}}/2$ over the choice of $i' \in I_{i^*,\omega}$ it holds that $w_1[i'] \neq C(x)[i']$. By Lemma 4.15, with probability at least $s_{\mathsf{code}} \cdot s_{\mathsf{pcp}} \cdot \rho_{\mathsf{code}}/4$, the local view of the recursive call to the corrector will be $\rho_{\mathsf{pcp}}/2$-far from any view that would make that recursive corrector return anything but $C(x)[i']$ or $\perp$. Recall that the corrector reads the value $w_1[i']$ q_{pcp} times, checks that they are all equal, and denotes it by η. Thus, $\eta = w_1[i']$ and moreover, the view of the recursive corrector is $\rho_{\mathsf{pcp}}/5$ from any view in which $\eta \neq w_1[i']$.

Since the *original* corrector (that is, the one that initiates the recursive call) compares η with the value given by the recursive invocation, with probability $(\rho_{\mathsf{code}}/2) \cdot (s_{\mathsf{code}} \cdot s_{\mathsf{pcp}} \cdot \rho_{\mathsf{code}}/4)$, its view is $\rho_{\mathsf{pcp}}/5$ far from any view that would make it output anything but $\perp$. □

Thus, in the rest of the analysis we may assume that $w_1|_{I_{i^*,\omega^*}}$ is $(\rho_{\mathsf{code}}/2)$-close to $C(x)|_{I_{i^*,\omega^*}}$.

Claim 4.23. *If* $w_1[i^*] \neq C(x)[i^*]$, *then* $\Pr[\mathsf{Good}] \geq s_{\mathsf{pcp}} \cdot \rho_{\mathsf{code}}/4$.

Proof. Consider the input (z_1, z_2) on which $\mathcal{V}_{i^*,\omega^*}$ is invoked. Recall that $z_1 = (w_1[i^*])^{q_{\mathsf{code}}}$ and $z_2 = w_1|_{I_{i^*,\omega^*}}$. By the strongly canonical soundness of the PCPP, with probability $s_{\mathsf{pcp}} \cdot \mu$, the local view of the PCPP verifier will be ρ_{pcp}-far from any view that would make it accept, where:

$$\mu = \min_{(z_1',z_2') \in S_{i^*,\omega^*}} \left\{ \max\left(\Delta\left((z_1,z_2),(z_1',z_2')\right), \Delta\left(\mathcal{P}(z_1',z_2'), \pi_{i^*,\omega^*}\right) \right) \right\}.$$

We show that $\mu \geq \rho_{\mathsf{code}}/4$. Let $(z_1', z_2') \in S_{i^*,\omega^*}$ that minimize μ. Then, there exists $\sigma \in \mathbb{F}$ such that $z_1' = \sigma^{q_{\mathsf{code}}}$ and $D_{i^*,\omega^*}(z_2') = \sigma$. If $\sigma \neq w_1[i^*]$, then $\mu \geq \Delta((z_1,z_2),(z_1',z_2')) \geq 1/2$, since the first part needs to be entirely changed. Thus, we may assume that $\sigma = w_1[i^*]$.

Assume for contradiction that z_2' is $(\rho_{\mathsf{code}}/2)$-close to $w_1|_{I_{i^*,\omega^*}}$. Then, by the triangle inequality z_2' is $\rho_{\mathsf{code}}/2 + \rho_{\mathsf{code}}/2 = \rho_{\mathsf{code}}$ close to $C(x)|_{I_{i^*,\omega^*}}$. However,

$$D_{i^*,\omega^*}(C(x)|_{I_{i^*,\omega^*}}) = C(x)[i^*] \neq w_1[i^*] = D_{i^*,\omega^*}(z_2').$$

Thus, by the first part of the robust RLCC definition, it must be the case that z_2' is ρ_{code}-far from $C(x)|_{I_{i^*,\omega^*}}$, which is a contradiction. Thus,

$$\mu \geq \Delta\left(z_2', z_2\right)/2 = \Delta\left(z_2', w_1|_{I_{i^*,\omega^*}}\right)/2 \geq \rho_{\mathsf{code}}/4.$$

So, with probability at least $s_{\mathsf{pcp}} \cdot \rho_{\mathsf{code}}/4$, the local view of the PCPP verifier will be ρ_{pcp} far from any view that would make it accept. Since the queries to the PCP consist of $1/5$ of the corrector's total queries, the claim follows. □

Thus, in the rest of the analysis we may assume also that $w_1[i^*] = C(x)[i^*]$.

Claim 4.24. *If* π_{i^*,ω^*} *is* ρ_{code}*-far from* $\mathcal{P}_{i^*,\omega^*}((C(x)[i^*])^{q_{\mathsf{code}}}, C(x)|_{I_{i^*,\omega^*}})$, *then* $\Pr[\mathsf{Good}] \geq s_{\mathsf{pcp}} \cdot \rho_{\mathsf{code}}/4$.

Proof. Consider yet again the input (z_1, z_2) on which $\mathcal{V}_{i^*,\omega^*}$ is invoked. Recall that $z_1 = (w_1[i^*])^{q_{\mathsf{code}}} = (C(x)[i^*])^{q_{\mathsf{code}}}$ and $z_2 = w_1|_{I_{i^*,\omega^*}}$. By the strongly canonical soundness of the PCPP, with probability

$s_{\mathsf{pcp}} \cdot \mu$, the local view of the PCPP verifier will be ρ_{pcp}-far from any view that would make it accept, where:

$$\mu = \min_{(z'_1,z'_2)\in S_{i^*,\omega^*}} \left\{ \max\left(\Delta\left((z_1,z_2),(z'_1,z'_2)\right), \Delta\left(\mathcal{P}(z'_1,z'_2),\pi_{i^*,\omega^*}\right) \right) \right\}.$$

We now show that $\mu \geq \rho_{\mathsf{code}}/4$. Suppose toward a contradiction that $\mu < \rho_{\mathsf{code}}/4$. Let $(z'_1,z'_2) \in S_{i^*,\omega^*}$ that minimize μ. That is,

$$\mu = \max\left(\Delta\left((z_1,z_2),(z'_1,z'_2)\right), \Delta\left(\mathcal{P}(z'_1,z'_2),\pi_{i^*,\omega^*}\right) \right). \tag{4.3}$$

First observe that if $z'_1 \neq (C(x)[i^*])^{q_{\mathsf{code}}}$, then $\mu \geq \Delta((z'_1,z'_2),(z_1,z_2)) \geq 1/2$. Thus, we may assume that $z'_1 = (C(x)[i^*])^{q_{\mathsf{code}}} = z_1$.

Equation (4.3) implies that $\mu \geq \Delta((z'_1,z'_2),(z_1,z_2))$ and so, $\Delta(z'_2,z_2) \leq 2\mu < \rho_{\mathsf{code}}/2$. But, by our assumption, $z_2 = w_1|_{I_{i^*,\omega^*}}$ is $\rho_{\mathsf{code}}/2$-close to $C(x)|_{I_{i^*,\omega^*}}$ and so, by the triangle inequality, $\Delta\left(z'_2, C(x)|_{I_{i^*,\omega^*}}\right) < \rho_{\mathsf{code}}$. By the first property of a robust RLCC, this implies that $z'_2 = C(x)|_{I_{i^*,\omega^*}}$.

Using Equation (4.3) yet again, we we obtain the following equation:

$$\mu \geq \Delta\left(\mathcal{P}(z'_1,z'_2),\pi_{i^*,\omega^*}\right) = \Delta\left(\mathcal{P}((C(x)[i^*])^{q_{\mathsf{code}}}, C(x)|_{I_{i^*,\omega^*}}),\pi_{i^*,\omega^*}\right)$$

which, by the claim's hypothesis is more than ρ_{code}. Thus, in any case $\mu \geq \rho_{\mathsf{code}}/4$.

Thus, with probability $s_{\mathsf{pcp}} \cdot \rho_{\mathsf{code}}/4$, the local view of the PCPP verifier will be ρ_{pcp}-far from any view that would make it accept. Since this view accounts for $1/5$ of the corrector's queries, the claim follows. $\square$

Hence, we may assume that π_{i^*,ω^*} is ρ_{code}-close to $\mathcal{P}_{i^*,\omega^*}((C(x)[i^*])^{q_{\mathsf{code}}}, C(x)|_{I_{i^*,\omega^*}})$.

Claim 4.25. $\Pr[\mathsf{Good}] \geq s_{\mathsf{pcp}}$.

Proof. The self-correction procedure of the PCPP is run on the string $\pi_{i^*,\omega}$, which is ρ_{code}-close to $\mathcal{P}_{i^*,\omega^*}((C(x)[i^*])^{q_{\mathsf{code}}}, C(x)|_{I_{i^*,\omega^*}})$. Since $\rho_{\mathsf{code}} \leq \delta_{\mathsf{pcp\text{-}correction\text{-}radius}}$, this means that the string is within the correcting radius of the PCPP. The claim follows by the robust local correction of the PCPP and the fact that these queries are a $1/5$ fraction of the queries that $\mathcal{M}'$ makes. $\square$

This concludes the proof of Lemma 4.21. $\square$

4.3 Exponential-length, constant-query, strongly canonical and self-correctable PCPP

In this section we construct self-correctable PCPPs for affine subspaces, with exponential length, constant query complexity, and strong soundness with respect to a canonical proof. Our construction is closely related to the Hadamard-based inner PCPP in [2].

Theorem 4.26. *Let $n \geq k$ be integers. Let $A \in \{0,1\}^{k\times n}$ be a matrix and let $b \in \{0,1\}^k$ be a vector. Then, the set $S_{A,b} = \{x \in \{0,1\}^n : Ax = b\}$ has a self-correctable, strongly canonical, $\Omega(1)$-robust* PCPP *with $O(1)$ queries and $O(n)$ randomness complexity, where the canonical proof string of the* PCPP *is a linear function of the input. Furthermore, the* PCPP *has a linear verifier.*

We note that, while the PCPP verifier in Theorem 4.26 is robust and checks a conjunction of linear functions of the symbols queried, we actually do not need these properties here. This is because these properties are used to ensure that resulting composed code behaves well under further composition, but this PCPP will be used in the *final* composition to construct our codes. We still prove these properties in order to formally use Theorem 4.13.

Proof. Fix a matrix $A \in \{0,1\}^{k \times n}$ and vector $b \in \{0,1\}^k$. We construct an (exponential length) self-correctable, strongly canonical PCPP for the set $S_{A,b}$, where the canonical proof string is a linear function of the input.

For a given input $x \in S_{A,b}$, the canonical PCPP proof string is the Hadamard encoding of the string x. Namely, the truth table of the function $\pi : \{0,1\}^n \to \{0,1\}$ defined as $\pi(z) = \langle z, x \rangle$, for every $z \in \{0,1\}^n$.

Given access to the input oracle $x \in \{0,1\}^n$ and an (alleged) proof oracle $\pi : \{0,1\}^n \to \{0,1\}$, the PCPP verifier $\mathcal{V}$ chooses at random $w, z \in \{0,1\}^n$ and performs the following tests:

- **Linearity Test:** check that $\pi(w) + \pi(z) = \pi(z+w)$ (note that the first addition refers to addition of scalars over $\{0,1\}$ whereas the second refers to vector addition over the vector space $\{0,1\}^n$).
- **Circuit Test:** choose at random $v \in \{0,1\}^k$ and check that $\pi(z) + \pi(z + vA) = \langle v, b \rangle$.
- **Input Proximity Test:** choose at random $i \in [n]$ and check that $\pi(z) + \pi(z + e_i) = x_i$, where $e_i \in \{0,1\}^n$ is the unit vector with 1 in its i-th coordinate and 0 everywhere else.

The verifier $\mathcal{V}$ accepts if and only if all of the tests above were successful. We proceed to show that this PCPP satisfies the required conditions.

Linearity. The canonical proof is the Hadamard encoding of the input, which is a linear code. The fact that the PCPP has a linear verifier follows because it checks a conjunction of linear constraints.

Self-correction. Follows immediately from the fact that the Hadamard code is a 2-query locally correctable code.

Robustness. Follows trivially from the fact that the number of queries is $O(1)$.

Completeness. Let $x \in S_{A,b}$ (i. e., $Ax = b$) and let π be the Hadamard encoding of x. Since π is a linear function, the linearity test passes with probability 1. For the circuit test observe that

$$\pi(z) + \pi(z + vA) = \langle z, x \rangle + \langle z + vA, x \rangle = \langle vA, x \rangle = \langle v, Ax \rangle = \langle v, b \rangle$$

as desired. For the input proximity test observe that

$$\pi(z) + \pi(z + e_i) = \langle z, x \rangle + \langle z + e_i, x \rangle = \langle e_i, x \rangle = x_i$$

which completes our analysis.

Strongly canonical soundness. Let $x \in \{0,1\}^n$ and fix a proof string $\widetilde{\pi}$. We need to show that $\mathcal{V}$ rejects with probability $\Omega(\delta)$, where

$$\delta \stackrel{\text{def}}{=} \min_{x' \in S_{A,b}} \left\{ \max\left(\Delta\left(x, x'\right), \Delta\left(P(x'), \widetilde{\pi}\right) \right) \right\}.$$

We assume without loss of generality[25] that $\delta \leq \frac{1}{2}$.

The proof goes as follows. We first use the celebrated linearity test of Blum, Luby and Rubinfeld [9] (see also [23]) to conclude that π must be very close to the Hadamard encoding of some string x'. Note that x' may be either close to (or even equal to) x or far from x. Moreover, using the local correctability of the Hadamard code, we can treat π as though it actually is *exactly* an Hadamard encoding of x'.

Suppose $x' \notin S_{A,b}$. Then, due to the distance of the Hadamard code, π is very far from the Hadamard encoding of any string $x^* \in S_{A,b}$, so the verifier needs to reject with constant probability. Ideally, the verifier would like to directly compute Ax' and compare it to b, but it only has access to the Hadamard code of x', not x' itself (and computing Ax' would require many queries). Similarly to the case of the Hadamard PCP, we observe that the requirement of $Ax' = b$ is a conjunction of k affine functions on x'. The high level idea is to take a random linear combination of these functions so that we end up with just a single linear function. More specifically, the verifier selects a random vector $v \in \{0,1\}^k$. We observe that if $Ax' = b$, then $\langle vA, x' \rangle = \langle v, b \rangle$, whereas if $Ax' \neq b$ then, with probability $1/2$, it holds that $\langle vA, x' \rangle \neq \langle v, b \rangle$. Thus, our verifier rejects if $\langle vA, x' \rangle \neq \langle v, b \rangle$ (where we observe that computing the inner product of x' with any fixed vector requires just a single query to the Hadamard encoding of x').

We are left with the case that $x' \in S_{A,b}$. In this case, we notice that x' is the minimizing string for δ and so $\delta = \Delta(x, x')$. So the verifier only needs to reject with probability proportional to $\Delta(x, x')$. To do this, it suffices to pick $i \in [n]$ at random, read the i^{th} bit of the input x, compare it to the i^{th} bit of x', and reject if they are different. We proceed to the formal proof.

The [9] test shows that if $\widetilde{\pi}$ is δ'-far from every linear function (for all sufficiently small $\delta' > 0$), then, with probability $\Omega(\delta')$, over the choice of z and w, it holds that $\widetilde{\pi}(z) + \widetilde{\pi}(w) \neq \widetilde{\pi}(z+w)$. Thus, we may assume without loss of generality that $\widetilde{\pi}$ is $\delta/4$-close to some linear function $\pi : \{0,1\}^n \to \{0,1\}$ (since otherwise the verifier rejects with probability $\Omega(\delta)$ when performing the linearity test). Let $x' \in \{0,1\}^n$ such that $\pi(z) = \langle z, x' \rangle$, for every $z \in \{0,1\}^n$.

Consider first the case that $x' \notin S_{A,b}$ (i. e., $Ax' \neq b$). In this case the verifier rejects in the circuit test

[25] Indeed, for fixed constant values of δ, the requirement that the verifier rejects with probability $\Omega(\delta)$ is trivial.

with probability:

$$\begin{aligned}
\Pr\left[\widetilde{\pi}(z)+\widetilde{\pi}(z+vA)=\langle v,b\rangle\right] &\geq \Pr\left[\pi(z)+\pi(z+vA)=\langle v,b\rangle\right]-2\cdot\delta/4 \\
&\geq \Pr\left[\langle z,x'\rangle+\langle z+vA,x'\rangle=\langle v,b\rangle\right]-\delta/2 \\
&= \Pr\left[\langle vA,x'\rangle=\langle v,b\rangle\right]-\delta/2 \\
&= \Pr\left[\langle v,Ax'+b\rangle=0\right]-\delta/2 \\
&= \frac{1}{2}-\delta/2 \\
&\geq \frac{1}{4}
\end{aligned}$$

where the first inequality is by the fact that z and $z+vA$ are each individually random in $\{0,1\}^n$ and our assumption that $\widetilde{\pi}$ is $\delta/4$-close to π, and the final equality uses the fact that $Ax'\neq b$.

Now consider the case that $x'\in S_{A,b}$ (i.e., $Ax'=b$). Note that by definition of δ, it holds that $\delta\leq\max\left(\Delta(x,x'),\Delta(P(x'),\widetilde{\pi})\right)\leq\max\left(\Delta(x,x'),\delta/4\right)$ and so $\Delta(x,x')\geq\delta$. Thus, the verifier rejects in the Input Proximity Test with probability at least

$$\begin{aligned}
\Pr\left[\widetilde{\pi}(z)+\widetilde{\pi}(z+e_i)\neq x_i\right] &\geq \Pr\left[\pi(z)+\pi(z+e_i)\neq x_i\right]-2\cdot\delta/4 \\
&= \Pr\left[\pi(e_i)\neq x_i\right]-\delta/2 \\
&= \Pr\left[x'_i\neq x_i\right]-\delta/2 \\
&= \Delta\left(x,x'\right)-\delta/2 \\
&\geq \delta/2.
\end{aligned}$$

□

4.4 Polynomial-length, polylog-query, strongly canonical, self-correctable and robust PCPP

In Section 4.3 we constructed a self-correctable, strongly canonical, robust PCPP with a constant number of queries and exponential size. That PCPP does not suffice for our end goal of an RLCC with polynomial block length and constant query.[26]

As described in the introduction, our first composition step will compose the low-degree extension code with a PCPP, which we call the "intermediate" PCPP. Thus, we want to construct a suitable PCPP. In this section we construct the intermediate PCPPs we use in our code construction. More formally, we prove the following theorem.

Theorem 4.27. *Let $A\in\{0,1\}^{k\times n}$ be a matrix and let $b\in\{0,1\}^k$ be a vector. There exist universal constants $c\in\mathbb{N}$ and $s,\rho\in(0,1)$ such that for every $m\leq\frac{\log(n)}{\log\log(n)}$, the set $S_{A,b}=\{x\in\{0,1\}^n : Ax=b\}$ has a* PCPP *(over the binary alphabet) with the following properties:*

[26]By composing that PCPP with a low-degree extension code (of suitable parameters), we obtain relaxed locally correctable codes with *quasi-polynomial* block length.

- *The* PCPP *is* ρ-robust *with strong soundness s (with respect to canonical proofs).*
- *The* PCPP *is* ρ-robust *self-correctable with soundness s.*
- *The canonical* PCPP *proof string is a* $\mathbb{GF}(2)$*-linear function of the input x. The length of the* PCPP *proof string is* $\mathrm{poly}(n)$.
- *The predicates that the* PCPP *verifier and self-corrector compute are also* $\mathbb{GF}(2)$ *linear functions. Both the verifier and the self correctors have query complexity* $O(m^3 n^{c/m})$ *and randomness complexity* $\mathrm{polylog}(n) \cdot n^{1/m}$.

Indeed, we will use this theorem where we set $m = \frac{\log(n)}{\log\log(n)}$, which yields a PCPP with polylogarithmic query complexity and polynomial length.[27]

Recall that a *strongly canonical* PCPP means that each valid input x has a unique accepting proof, and the verifier must reject any other proof with probability proportional to how far away the proof is from the correct one. Additionally, self-correctability refers to the fact that the canonical proofs form a locally correctable code, while robustness stipulates that the view of the verifier on any invalid input x must be far from an accepting view. See Section 4.1 for details.

We note that the proof of Lemma 4.28 is closely related to (and influenced by) the construction of a linear inner proof system (LIPS) in [27, Theorem 5.19]. The following lemma is the main step in proving Theorem 4.27.

Lemma 4.28. Let $A \in \{0,1\}^{k\times n}$ be a matrix and let $b \in \{0,1\}^k$ be a vector. For every $m \le \frac{\log(n)}{\log\log(n)}$ and for every finite field $\mathbb{F}$ of characteristic 2 and order $|\mathbb{F}| \ge \Omega(m\cdot n^{1/m})$, the set $S_{A,b} = \{x \in \{0,1\}^n : Ax = b\}$ has a strongly canonical PCPP over the alphabet $\mathbb{F}$, and has query complexity $O(m^2 \cdot n^{2/m})$ and randomness complexity $O(m^2 \cdot n^{2/m} \cdot \log(|\mathbb{F}|))$.

Furthermore, (1) the canonical PCPP proof string is an $O(m)$-variate total degree $O(m\cdot n^{1/m})$ polynomial, (2) the canonical PCPP proof string viewed as a bit string (in the natural way), is a $\mathbb{GF}(2)$-linear function of the input, (3) the PCPP verifier, viewed as a function on bits, computes a $\mathbb{GF}(2)$ affine relation on its view, and (4) the PCPP verifier only makes a single query to the input.

Indeed, this lemma gives the desired PCPP for Theorem 4.27, modulo the self-correction and robustness properties.

In Section 4.4.1 we prove Lemma 4.28 and then in Section 4.4.2 we use Lemma 4.28 to prove Theorem 4.27.

4.4.1 Proof of Lemma 4.28

Fix $A \in \{0,1\}^{k\times n}$ and $b \in \{0,1\}^k$. We construct a strongly canonical self-correctable PCPP for verifying membership in $S_{A,b}$. Let $C : \{0,1\}^n \to \{0,1\}^k$ be a circuit composed only of fan-in 2 parity gates such

[27] We believe that with more care our approach would yield a PCP with only logarithmic randomness (rather than polylogarithmic). However, we do not optimize the randomness complexity since it is not required for our main results.

that on input $x \in \{0,1\}^n$ it outputs $y \in \{0,1\}^k$ where y is the all zeros string if and only if $x \in S_{A,b}$.[28] We denote the size (i. e., number of wires) of C by N and note that $N = O(n \cdot k)$.

Let $\mathbb{F}$ be a field of characteristic 2, let $\mathbb{H} \subseteq \mathbb{F}$ an arbitrary subset such that $\{0,1\} \subseteq \mathbb{H}$, and where $|\mathbb{H}| = \lceil N^{1/m} \rceil$ and $|\mathbb{F}| = \Omega(|\mathbb{H}| \cdot m)$. Note that $|\mathbb{H}^m| \geq N$. We associate the integers in $[N]$ with the first N elements in $\mathbb{H}^m$, given in lexicographic order.

For a given input $x \in \{0,1\}^n$, we extend x to be a string in $\{0,1\}^N$ by associating the i-th entry with the value of the i-th wire of C on input x. We list the wires in order of increasing depth—this puts the input x in the first n positions of the extended string and the output $C(x)$ in the last k positions.

Let $\ell = 4m+4$. We define a polynomial $X : \mathbb{F}^\ell \to \mathbb{F}$ to be the (unique) individual degree $|\mathbb{H}|-1$ polynomial such that for every $i \in [N]$, it holds that $X(i) = x_i$ and for every $i \in \mathbb{H}^\ell \setminus [N]$ it holds that $X(i) = 0$.[29]

We define a function $\phi_{A,b} : (\mathbb{H}^m)^3 \times (\{0,1\})^4 \to \{0,1\}$ such that for every $i_1, i_2, i_3 \in \mathbb{H}^m$ and $\alpha_1, \alpha_2, \alpha_3$, $\sigma \in \{0,1\}$ we define $\phi_{A,b}(i_1, i_2, i_3, \alpha_1, \alpha_2, \alpha_3, \sigma) = 1$ if there is a gate of the form

$$\alpha_1 \cdot x_{i_1} + \alpha_2 \cdot x_{i_2} + \alpha_3 \cdot x_{i_3} + \sigma = 0$$

in the circuit and $\phi_{A,b}(i_1, i_2, i_3, \alpha_1, \alpha_2, \alpha_3, \sigma) = 0$ otherwise. We first extend $\phi_{A,b}$ to be a function $\phi_{A,b}$: $\mathbb{H}^{3m+4} \to \{0,1\}$ by defining it to be zero outside of $\mathbb{H}^{3m} \times \{0,1\}^4$. Let $\hat{\phi}_{A,b} : \mathbb{F}^{3m+4} \to \mathbb{F}$ be the low-degree extension of $\phi_{A,b}$.

The idea behind the definition of $\phi_{A,b}$ is that it encodes local constraints, that, put together, encode the *global* correctness of a computation performed by the circuit C. Namely, $\phi_{A,b}$ is nonzero only at tuples that encode gates present in C, and is zero otherwise. Suppose a prover gives the verifier a polynomial that the prover claims is 0 at all tuples that encode gates in C. The verifier can check this claim by multiplying that polynomial with $\hat{\phi}_{A,b}$ and checking that the product is identically 0. This property will be particularly useful if we create a polynomial that is 0 at all tuples encoding gates in C if and only if the assignment to the wires satisfies the relations imposed by the gates. In this case, we will be able to check that the assignment that the prover gives is valid via polynomial identity testing on this product polynomial. This motivates our definition of the polynomial P_0 below.

Recall that $\ell = 4m+4$. We define a polynomial $P_0 : \mathbb{F}^\ell \to \mathbb{F}$ such that for every $z \in \mathbb{F}^\ell$, where $z = (i_1, i_2, i_3, i_4, \alpha_1, \alpha_2, \alpha_3, \sigma) \in (\mathbb{F}^m)^4 \times \mathbb{F}^4$, the following equation holds.

$$\begin{aligned} P_0(z) = &\hat{\phi}_{A,b}(i_1, i_2, i_3, \alpha_1, \alpha_2, \alpha_3, \sigma) \cdot \left(\alpha_1 \cdot X\left(0^{\ell-m}, i_1\right) + \alpha_2 \cdot X\left(0^{\ell-m}, i_2\right) + \alpha_3 \cdot X\left(0^{\ell-m}, i_3\right) + \sigma\right) \\ &+ X\left(0^{\ell-m}, i_4\right) \cdot \left(1 - X\left(0^{\ell-m}, i_4\right)\right). \end{aligned}$$

Notice that P_0 is the sum of two terms. The first term is exactly what we described beforehand: a product of $\hat{\phi}_{A,b}$ with a simple polynomial that checks that the assignment to the wires satisfies each gate of the circuit C. The purpose of the second term is to *enforce booleanity* in the input. Namely, $X\left(0^{\ell-m}, i_4\right) \cdot \left(1 - X\left(0^{\ell-m}, i_4\right)\right)$ will be nonzero whenever $X\left(0^{\ell-m}, i_4\right)$ is non-boolean, so this means that

[28] For each coordinate $i \in [k]$, we can compute $\sum_j A_{ij} x_j + b_i$ mod 2 with at most $n+1$ parity gates with 0/1 weights. It will be convenient to include wires with 0-weights for notation purposes. We can do this for all coordinates i to get the desired circuit.

[29] The reader may find it mysterious why X was defined as an ℓ variate polynomial rather than m-variate (since we are essentially taking the low-degree extension of the computation which is of size $N \leq |\mathbb{H}|^m$. Jumping ahead, we note that the reason that we do so is to facilitate the "bundling" of this polynomial with other ℓ-variate polynomials that are defined below.

P_0 is identically 0 on $\mathbb{H}^\ell$ only if X encodes a boolean assignment x such that $C(x) = 0$. This will be shown formally in Theorem 4.31.

Thus, we would like to be able to check that P_0 is identically 0 over $\mathbb{H}^\ell$. Doing so is not immediate since P_0 has degree $O(m \cdot |\mathbb{H}|)$ and so it does not necessarily have to be identically 0 on all of $\mathbb{F}^\ell$ (indeed, typically it will not be). As usual in the PCP and interactive proof literature, we enforce this condition using the sumcheck approach of Lund *et al.* [39].

The key idea is to define additional auxiliary polynomials $P_1, \ldots, P_\ell : \mathbb{F}^\ell \to \mathbb{F}$ (which are often called the sumcheck polynomials) and are defined as follows. For every $(z_1, \ldots, z_\ell) \in \mathbb{F}^\ell$:

$$P_i(z_1, \ldots, z_\ell) = \sum_h P_{i-1}(z_1, \ldots, z_{i-1}, h, z_{i+1}, \ldots, z_\ell) \cdot z_i^h, \tag{4.4}$$

where by exponentiating in h, we mean that we associate $\mathbb{H}$ with the integers $\{0, \ldots, |\mathbb{H}| - 1\}$ in some canonical way, and simply exponentiate by the corresponding integer. Notice that Equation (4.4) can be equivalently rewritten as:

$$P_i(z_1, \ldots, z_\ell) = \sum_{h_1, \ldots, h_i} P_0(h_1, \ldots, h_i, z_{i+1}, \ldots, z_\ell) \cdot z_1^{h_1} \cdot \cdots \cdot z_i^{h_i}. \tag{4.5}$$

The PCPP proof string. The PCP proof bundles the polynomials $X, \mathcal{P}_0, \ldots, P_\ell$ in the following way. Let $\lambda, \lambda_0, \ldots, \lambda_\ell$ be distinct elements in $\mathbb{F}$. We define a polynomial $Q : \mathbb{F}^{\ell+1} \to \mathbb{F}$ as follows. For every $z \in \mathbb{F}^\ell$ we define $Q(\lambda, z) = X(z)$ and $Q(\lambda_i, z) = P_i(z)$, for every $i \in \{0, \ldots, \ell\}$. We extend the definition of Q to $\mathbb{F}^{\ell+1}$ by interpolation (in the first variable).

Observe that Q has degree $\ell + 1$ in its first variable and total degree $O(|\mathbb{H}| \cdot m)$.

The PCPP verifier. Recall that the PCPP verifier is given access to two oracles, the input oracle $x \in \{0,1\}^n$ and an alleged proof oracle $Q : \mathbb{F}^{\ell+1} \to \mathbb{F}$. The verifier first runs a total degree test (see Lemma 4.10) on Q, with respect to degree $O(m \cdot |\mathbb{H}|)$. If the test fails the verifier immediately rejects.

We "un-bundle" the polynomial $Q : \mathbb{F}^{\ell+1} \to \mathbb{F}$ as follows. Let $X : \mathbb{F}^\ell \to \mathbb{F}$ and $P_0, \ldots, P_\ell$ be defined as $X(z) = Q(\lambda, z)$ and $P_i(z) = Q(\lambda_i, z)$, where $\lambda, \lambda_0, \ldots, \lambda_\ell \in \mathbb{F}$ are the fixed field elements as defined above. In the sequel, whenever we say that the verifier queries one of the polynomials Q, X, $P_0, \ldots, P_\ell$, we actually mean that it reads these values via the self-correction procedure (see Lemma 4.9) applied to Q (with respect to degree $O(m \cdot |\mathbb{H}|)$).

The verifier runs the following additional tests. If any test fails then the verifier rejects. Otherwise, it accepts.

1. **Individual Degree Test for First Variable of Q:** The verifier chooses at random $(z_1, z_2, \ldots, z_{\ell+1}) \in \mathbb{F}^{\ell+1}$. It checks that the value $Q(z_1, \ldots, z_{\ell+1})$ is consistent with value obtained by interpolating from $Q(\gamma, z_2, \ldots, z_{\ell+1})$ and $\{Q(\gamma_i, z_2, \ldots, z_{\ell+1})\}_{i \in \{0, \ldots, \ell\}}$ (where we recall that all values are read via self-correction from Q).

2. **Low Individual Degree $|\mathbb{H}| - 1$ Test for X.** The verifier runs the individual degree $|\mathbb{H}| - 1$ test on X (see Lemma 4.11).

3. **Zero Output (and Padding) for X:** check that the $|\mathbb{H}|^{\ell} - N + k$ bit suffix of $X|_{\mathbb{H}^{\ell}}$ is the all zero string. An efficient procedure for doing so is described in, e. g., [44, Proposition 3.9].

4. **Input Proximity Test:** choose at random $i \in [n]$ and check that $X(0^{\ell-m}, i) = x_i$. (Recall that we use the local correction procedure for this.)

5. X **vs.** P_0 **Test:** choose at random $z \in \mathbb{F}^{\ell}$. Let $i_1, i_2, i_3, i_4 \in \mathbb{F}^m$ and $\alpha_1, \alpha_2, \alpha_3, \sigma \in \mathbb{F}$ such that $z = (i_1, i_2, i_3, i_4, \alpha_1, \alpha_2, \alpha_3, \sigma)$. Check that
$$P_0(z) = \hat{\phi}_{A,b}(z) \cdot \left(\alpha_1 \cdot X(0^{\ell-m}, i_1) + \alpha_2 \cdot X(0^{\ell-m}, i_2) + \alpha_3 \cdot X(0^{\ell-m}, i_3) + \sigma\right)$$
$$+ X\left(0^{\ell-m}, i_4\right) \cdot \left(1 - X\left(0^{\ell-m}, i_4\right)\right).$$

6. **Sumcheck Test:** choose at random $(z_1, \ldots, z_\ell) \in \mathbb{F}^{\ell}$ and check that for every $i \in [\ell]$:
$$P_i(z_1, \ldots, z_\ell) = \sum_{h_i \in \mathbb{H}} P_{i-1}(z_1, \ldots, z_{i-1}, h_i, z_{i+1}, \ldots, z_\ell) \cdot z_i^{h_i},$$
where the values $\{P_{i-1}(z_1, \ldots, z_{i-1}, h, z_{i+1}, \ldots, z_\ell)\}_{h \in \mathbb{H}}$ are read via the self-correction procedure.

7. P_ℓ **Zero Test:** choose random $z \in \mathbb{F}^{\ell}$ and check that $P_\ell(z) = 0$.

Query complexity. It can be readily verified that each one of our tests makes at most $O(m \cdot |\mathbb{H}|)$ calls to the self correction procedure. The latter procedure requires $O(m|\mathbb{H}|)$ queries and so we obtain query complexity $O(m^2 \cdot |\mathbb{H}|^2)$.

Randomness complexity. It can also be readily verified that each of our tests by itself uses $O(\log(|\mathbb{F}|^m))$ randomness. In addition, for each one of queries we invoke the self-correction procedure which induces uses $O(\log(|\mathbb{F}|^m))$ randomness. Thus, the total amount of randomness used is $O(m \cdot |\mathbb{H}| \cdot \log(|\mathbb{F}|^m))$.

Completeness. Suppose that we have an input $x \in \{0,1\}^n$ such that $C(x) = 0$, and that we construct the polynomials X, $P_1, \ldots P_\ell$ from this input and bundle them into Q as specified above.

Clearly the Zero Suffix test passes, since this is how we constructed the polynomial. The input proximity test passes because $X(0^{\ell-m}, i) = x_i$ for all i by construction. The X vs. P_0 test passes because P_0 is constructed correctly from X (and in particular, X is boolean valued inside $\mathbb{H}^{\ell}$). The sumcheck test passes because P_{i+1} is constructed correctly from P_i.

Only the fact that the Zero Test for P_ℓ passes remains to be analyzed. Since x is a boolean string such that $C(x) = 0$, we conclude that $P_0|_{\mathbb{H}^{\ell}} \equiv 0$. Now we establish that $P_i|_{\mathbb{F}^i \times \mathbb{H}^{\ell-i}} \equiv 0$ for all $i \in [\ell]$. Assume that the claim is true for $i-1$. Fix a point $\vec{z} = (z_1, \ldots, z_\ell) \in \mathbb{F}^i \times \mathbb{H}^{\ell-i}$. By definition, we have that
$$P_i(z_1, \ldots, z_\ell) = \sum_{h_i \in \mathbb{H}} P_{i-1}(z_1, \ldots, z_{i-1}, h_i, z_{i+1}, \ldots, z_\ell) \cdot z_i^{h_i}.$$
Applying the inductive hypothesis, we see that, for all $h_i \in \mathbb{H}$,
$$P_{i-1}(z_1, \ldots, z_{i-1}, h_i, z_{i+1}, \ldots, z_\ell) \cdot z_i^{h_i}|_{\mathbb{F}^i \times \mathbb{H}^{\ell-i}} \equiv 0.$$

Hence, by induction, we have that $P_\ell|_{\mathbb{F}^{\ell}} \equiv 0$, and so the Zero Test passes.

Strongly canonical soundness. Let $x \in \{0,1\}^n$ and fix a proof string $\widetilde{Q} : \mathbb{F}^{\ell+1} \to \mathbb{F}$. We need to show that $\mathcal{V}$ rejects with probability $\Omega(\delta)$, where

$$\delta \stackrel{\text{def}}{=} \min_{x' \in S_{A,b}} \left\{ \max\left(\Delta\left(x, x'\right), \Delta\left(Q(x'), \widetilde{Q}\right) \right) \right\}, \tag{4.6}$$

where $Q(x')$ denotes the canonical PCPP proof string for input x'.

We assume without loss of generality that $\widetilde{Q}$ is δ-close to a total degree $O(|\mathbb{H}| \cdot m)$ polynomial Q^*, since otherwise the total degree test fails with high probability.

Claim 4.29 (Individual Degree of Q in its First Variable)**.** *If Q^* does not have individual degree $\ell+1$ in its first variable, then the verifier rejects with probability $1 - \frac{O(|\mathbb{H}|\cdot\ell)}{|\mathbb{F}|}$.*

Proof. Suppose that Q^* has individual degree $t > \ell+1$ in its first variable (but not $t-1$). We can write Q^* as:

$$Q^*(z_1, \ldots, z_{\ell+1}) = \sum_{i=0}^{t} z_1^i \cdot Q_i(z_2, \ldots, z_{\ell+1})$$

for some total degree $O(|\mathbb{H}| \cdot m)$ polynomials $Q_0, \ldots, Q_t : \mathbb{F}^\ell \to \mathbb{F}$. Furthermore, the polynomial Q_t is not identically zero (since otherwise Q^* would have individual degree $t-1$ in z_1). Thus, with probability $1 - O(|\mathbb{H}| \cdot \ell)/|\mathbb{F}|$, it holds that $Q_t(z_2, \ldots, z_{\ell+1}) \neq 0$. Assuming that the latter holds, the univariate polynomial $T(z_1) = \sum_{i=0}^{t} z_1^i \cdot Q_i(z_2, \ldots, z_{\ell+1})$ has degree t but not degree $t-1$. Therefore with probability $1 - O(\ell)/|\mathbb{F}|$ the value $T(z_1)$ is inconsistent with the value obtained by interpolating from $T(\gamma)$ and $\{T(\gamma_i)\}_{i \in \{0,\ldots,\ell\}}$. Therefore, our verifier rejects with probability $1 - O(|\mathbb{H}| \cdot \ell)/|\mathbb{F}|$. □

Hence, we can assume that Q^* has individual degree $\ell+1$ in its first variable. Let $X^* : \mathbb{F}^\ell \to \mathbb{F}$ and $P_0^*, \ldots, P_\ell^* : \mathbb{F}^\ell \to \mathbb{F}$ be defined as $X^*(z) = Q^*(\gamma, z)$ and $P_i^*(z) = Q^*(\gamma_i, z)$, for every $z \in \mathbb{F}^\ell$ and $i \in \{0, \ldots, \ell\}$. Since the verifier accesses $\widetilde{Q}$ using the self-correction procedure , we can assume that the verifier has direct access to the polynomials $X^*, P_0^*, \ldots, P_\ell^*$. This assumption is without loss of generality, as we can make the error probability of the self-correction procedure sufficiently low such that an error in reading any of the polynomials $X^*, P_0^*, \ldots, P_\ell^*$ only occurs with negligible probability, with a negligible increase in query complexity.

We now give some high-level intuition for the proof of strongly canonical soundness. We want to show that at least one of the tests will reject with probability $\Omega(\delta)$. We will think about this in the contrapositive: we want to show that, if the test accepts with probability $> 1 - O(\delta)$, then the proof strings $X^*, P_0^*, \ldots, P_\ell^*$ are very close to the canonical proofs for a bit string $x' \in S_{A,b}$, and furthermore that $\Delta(x, x') < \delta$.

We will start by assuming that the "P_ℓ Zero Test" accepts with high probability, and use the acceptance of sumcheck tests to conclude that $P_0^*|_{\mathbb{H}^\ell} \equiv 0$ (Theorem 4.30). This argument is similar to the proof we gave that the P_ℓ Zero Test accepts in the Completeness case. Additionally, the acceptance of the X vs. P_0 Test and the Zero Suffix Test tells us that P_0^* was appropriately constructed from X^*. These two facts together let us conclude that the first n elements of X^* encode a boolean string x' such that $x' \in S_{A,b}$, and that X^* is close to a polynomial X' that is a part of the canonical proof generated with respect to x' (Theorem 4.32). The sumcheck tests let us extend this observation to show that *all* the proof polynomials

$X^*, P_0^*, \ldots, P_\ell^*$ are very close to the canonical polynomials generated with respect to x' (Theorem 4.34). Finally, we verify that x' is close to the input x using the Input Proximity Test (Theorem 4.36).

Claim 4.30. *If for some $i \in \{0, \ldots, \ell\}$ it holds that $P_i^*|_{\mathbb{F}^i \times \mathbb{H}^{\ell-i}} \not\equiv 0$, then the verifier rejects with probability $1 - \frac{O(|\mathbb{H}| \cdot \ell)}{|\mathbb{F}|}$.*

Proof. We proceed by reverse induction on i. For $i = \ell$ the claim follows from the zero test that the verifier runs on P_ℓ^*, and the Polynomial Identity Lemma (Lemma 4.8). For the inductive step, we will rely on the sumcheck test.

Assume that the claim holds for $i \in [\ell]$. We show that it holds for $i-1$. We may assume that $P_i^*|_{\mathbb{F}^i \times \mathbb{H}^{\ell-i}} \equiv 0$ (since otherwise the claim follows from the inductive hypothesis). Suppose that the restriction $P_{i-1}^*|_{\mathbb{F}^{i-1} \times \mathbb{H}^{\ell-i+1}}$ is not identically zero. We need to show that the verifier rejects with high probability.

Define $T_i(z_1, \ldots, z_\ell) = \sum_{h_i \in \mathbb{H}} P_{i-1}^*(z_1, \ldots, z_{i-1}, h_i, z_{i+1}, \ldots, z_\ell) \cdot z_i^{h_i}$. Observe that by our assumption that $P_{i-1}^*|_{\mathbb{F}^{i-1} \times \mathbb{H}^{\ell-i+1}} \not\equiv 0$ it follows that $T|_{\mathbb{F}^i \times \mathbb{H}^{\ell-i}} \not\equiv 0$ and in particular it must be distinct from P_i^*. Since both have total degree $O(m \cdot |\mathbb{H}|)$, by the Polynomial Identity Lemma (Lemma 4.8), with probability $1 - O(m \cdot |\mathbb{H}|/|\mathbb{F}|)$ over $z_1, \ldots, z_\ell \in \mathbb{F}$ it holds that

$$P_i^*(z_1, \ldots, z_\ell) \neq T_i(z_1, \ldots, z_\ell) = \sum_{h_i \in \mathbb{H}} P_{i-1}^*(z_1, \ldots, z_{i-1}, h_i, z_{i+1}, \ldots, z_\ell) \cdot z_i^{h_i},$$

in which case the verifier rejects in the sumcheck test. □

In particular, by Theorem 4.30 (with $i = 0$), we may assume that $P_0^*|_{\mathbb{H}^\ell} \equiv 0$. Let $x' \in \mathbb{F}^n$ be the restriction of X^* to the first n field elements. (Recall that these refer to the input bits.) Let $Q(x') = (X', P_0', \ldots, P_\ell')$ be the canonical PCPP proof with respect to the input x'.[30]

Claim 4.31. *If $x' \notin \{0,1\}^n$ (i. e., x' is not Boolean valued) then, with probability $1 - O(m \cdot |\mathbb{H}|/|\mathbb{F}|)$, the verifier rejects.*

Proof. Suppose that $x'_{i^*} \notin \{0,1\}$ for some $i^* \in [n]$. In particular this means that $x'_{i^*} \cdot (1 - x'_{i^*}) \neq 0$ (since the polynomial $\lambda \cdot (1-\lambda)$ has precisely two solutions: 0 and 1).

Define a polynomial $T_0 : \mathbb{F}^\ell \to \mathbb{F}$ as $T_0(i_1, i_2, i_3, i_4, \alpha_1, \alpha_2, \alpha_3, \sigma) = \hat{\phi}_{A,b}(i_1, i_2, i_3, \alpha_1, \alpha_2, \alpha_3, \sigma) \cdot \big(\alpha_1 \cdot X^*(i_1) + \alpha_2 \cdot X^*(i_2) + \alpha_3 \cdot X^*(i_3) + \sigma\big) + X^*(i_4) \cdot (1 - X^*(i_4))$. Observe that $T_0(\vec{0}, \vec{0}, \vec{0}, i^*, 0, 0, 0, 0) = X^*(i^*) \cdot (1 - X^*(i^*)) \neq 0$ whereas $P_0^*(\vec{0}, \vec{0}, \vec{0}, i^*, 0, 0, 0, 0) = 0$ (since we have assumed that it is identically 0 in $\mathbb{H}^\ell$). Thus, by the Polynomial Identity Lemma (Lemma 4.8), with probability $1 - O(m \cdot |\mathbb{H}|/|\mathbb{F}|)$ over the choice of $z \in \mathbb{F}^\ell$, it holds that $P^*(z) \neq T(z)$, in which case the verifier rejects. □

Thus, we may assume that $x' \in \{0,1\}^n$.

Claim 4.32. *If $X^* \not\equiv X'$, then the verifier rejects with probability $1 - O(m \cdot |\mathbb{H}|/|\mathbb{F}|)$.*

[30] Note that we have not yet established that x' is Boolean valued (this fact will be established in Theorem 4.31), nor that $x \in S_{A,b}$. Nevertheless, at least syntactically, the construction of the canonical PCPP proof string given above extends naturally for every input in $\mathbb{F}^n$.

Proof. We will first show that if $X^*|_{\mathbb{H}^\ell} \not\equiv X'|_{\mathbb{H}^\ell}$ then the verifier rejects with high probability.

Suppose that there exists an index $i \in \{N+1,\dots,\mathbb{H}^\ell\}$ such that $X^*(i) \neq X'(i)$. This means that X^* and X' differ on the "zero region." Since X' was defined as a canonical polynomial for input x', we know that X' is 0 on the "zero region" by construction, which means that $X^*(i) \neq 0$. In this case, the verifier rejects with probability $1-O(m\cdot|\mathbb{H}|/|\mathbb{F}|)$ due to the Zero Output and Padding Test (see the analysis of this test in [44, Proposition 3.9]).

Now, we can assume that the suffixes match, so if $X^*|_{\mathbb{H}^\ell} \not\equiv X'_{\mathbb{H}^\ell}$, they must differ on some index in $[N]$. Let $i_1^* \in [N]$ be the smallest such that $X^*(i_1^*) \neq X'(i_1^*)$ (notice that $i_1^* > n$ since we have defined x' as the n element prefix of X^*). Since the circuit is deterministic, the value at each wire is determined by the input x' to the circuit. By definition, the correct value of each wire is included in X'. Since our assumption is that $X^*(i_1^*) \neq X'(i_1^*)$, there exist $\alpha_1^*,\alpha_2^*,\alpha_3^*,\sigma \in \{0,1\}$ and $i_2^*,i_3^* \in \mathbb{H}^m$ such that $\phi_{A,b}(i_1^*,i_2^*,i_3^*,\alpha_1^*,\alpha_2^*,\alpha_3^*,\sigma)=1$ but $\alpha_1^*\cdot X^*(i_1^*)+\alpha_2^*\cdot X^*(i_2^*)+\alpha_3^*\cdot X^*(i_3^*)+\sigma \neq 0$.

Define a polynomial $T_0 : \mathbb{F}^\ell \to \mathbb{F}$ as follows. For every $z \in \mathbb{F}^\ell$, where $z=(i_1,i_2,i_3,i_4,\alpha_1,\alpha_2,\alpha_3,\sigma) \in (\mathbb{F}^m)^4 \times \mathbb{F}^4$ it holds that

$$\begin{aligned} T_0(z) =& \hat{\phi}_{A,b}\big(i_1,i_2,i_3,\alpha_1,\alpha_2,\alpha_3,\sigma\big)\cdot\Big(\alpha_1\cdot X^*\big(0^{\ell-m},i_1\big)+\alpha_2\cdot X^*\big(0^{\ell-m},i_2\big)+\alpha_3\cdot X^*\big(0^{\ell-m},i_3\big)+\sigma\Big) \\ &+X^*\big(0^{\ell-m},i_4\big)\cdot\Big(1-X^*\big(0^{\ell-m},i_4\big)\Big). \end{aligned}$$

Observe that $T_0(i_1^*,i_2^*,i_3^*,\vec{0},\alpha_1^*,\alpha_2^*,\alpha_3^*,\sigma^*) \neq 0$ whereas $P_0^*(i_1^*,i_2^*,i_3^*,\vec{0},\alpha_1^*,\alpha_2^*,\alpha_3^*,\sigma^*)=0$ (since we have assumed that P_0^* is identically 0 in $\mathbb{H}^\ell$). Thus, with probability $1-O(m\cdot|\mathbb{H}|/|\mathbb{F}|)$ over the choice of $z \in \mathbb{F}^\ell$ it holds that $P_0^*(z) \neq T_0(z)$, in which case the verifier rejects on the X vs. P_0 Test.

Hence, we may assume that $X'|_{\mathbb{H}^\ell} \equiv X^*|_{\mathbb{H}^\ell}$. By construction X' has individual degree $|\mathbb{H}|-1$. If X^* does not have individual degree $|\mathbb{H}|-1$, then the individual degree $|\mathbb{H}|-1$ rejects with probability $1-O(m\cdot|\mathbb{H}|)/|\mathbb{F}|$. Hence, we may assume that also X^* has individual degree $|\mathbb{H}|-1$. Two individual degree $|\mathbb{H}|-1$ polynomials that agree on $\mathbb{H}^\ell$ must agree on $\mathbb{F}^\ell$ and so we obtain that $X^* \equiv X'$. □

Thus we may assume that $X^* \equiv X'$. Now we can use the X vs. P_0 Test again to argue that the verifier will reject if $P_0' \not\equiv P_0^*$.

Claim 4.33. *If $P_0' \not\equiv P_0^*$ then the verifier rejects with probability* $1-O(|\mathbb{H}|\cdot m/|\mathbb{F}|)$.

Proof. Since we have assumed that $X^* \equiv X'$ (due to Theorem 4.32), we know that

$$\begin{aligned} P_0'(z) =& \hat{\phi}_{A,b}(i_1,i_2,i_3,\alpha_1,\alpha_2,\alpha_3,\sigma)\cdot\big(\alpha_1\cdot X^*(0^{\ell-m},i_1)+\alpha_2\cdot X^*(0^{\ell-m},i_2)+\alpha_3\cdot X^*(0^{\ell-m},i_3)+\sigma\big) \\ &+X^*(0^{\ell-m},i_4)\cdot(1-X^*(0^{\ell-m},i_4)). \end{aligned}$$

So we can rephrase the X vs. P_0 Test as testing whether P_0^* is equal to P_0' at a random point. Since P_0^* and P_0' are both polynomials of degree at most $O(m\cdot|\mathbb{H}|)$, we can conclude that, if P_0' and P_0^* are not equivalent, the test will reject with probability $1-O(m\cdot|\mathbb{H}|/|\mathbb{F}|)$. □

Now we can assume that $P_0' \equiv P_0^*$. We will use the Sumcheck Tests to conclude that, if P_i' and P_i^* are not equivalent for any $i \in [\ell]$, the test will reject with probability $1-O(m\cdot|\mathbb{H}|/|\mathbb{F}|)$.

Claim 4.34. *For every $i \in \{0,\dots,\ell\}$, if $P_i^* \not\equiv P_i'$, then the verifier rejects with probability $1 - O(m \cdot |\mathbb{H}|/|\mathbb{F}|)$.*

Proof. We prove by induction on i. The base case $i = 0$ follows from Theorem 4.33. Assume that the claim holds for $i-1$ and we show that it holds for i. We may assume that $P_{i-1}^* \equiv P_{i-1}'$ (since otherwise the claim follows from the inductive hypothesis).

Suppose that $P_i^* \not\equiv P_i'$. Thus, by the Polynomial Identity Lemma (Lemma 4.8), the following holds with probability $1 - O(m \cdot |\mathbb{H}|/|\mathbb{F}|)$ over the choice of $z_1,\dots,z_\ell \in \mathbb{F}$:

$$\begin{aligned} P_i^*(z_1,\dots,z_\ell) \neq P_i'(z_1,\dots,z_\ell) \\ &= \sum_{h_i \in \mathbb{H}} P_{i-1}'(z_1,\dots,z_{i-1},h_i,z_{i+1},\dots,z_\ell) \cdot z_i^{h_i} \\ &= \sum_{h_i \in \mathbb{H}} P_{i-1}^*(z_1,\dots,z_{i-1},h_i,z_{i+1},\dots,z_\ell) \cdot z_i^{h_i}, \end{aligned}$$

in which case the verifier rejects. □

Thus, we may assume that $P_i' \equiv P_i^*$ for all $i \in \{0,\dots,\ell\}$.

So we have established that $Q^*(\gamma,\cdot,\dots,\cdot) \equiv Q'(\gamma,\cdot,\dots,\cdot)$ and $Q^*(\gamma_i,\cdot,\dots,\cdot) \equiv Q'(\gamma_i,\cdot,\dots,\cdot)$, for all $i \in [\ell]$. Since Q' has individual degree $\ell+1$ by construction and Q^* by our assumption, they must agree on $\mathbb{F}^{\ell+1}$. Thus, $Q' \equiv Q^*$.

Claim 4.35. *If $x' \notin S_{A,b}$ then the verifier rejects with probability $1 - \frac{O(m\cdot|\mathbb{H}|)}{|\mathbb{F}|}$.*

Proof. Since $x' \notin S_{A,b}$, we know that $C(x') \neq 0$. This means that there exists an index $N-k \le i \le N$ such that $X'(i) \neq 0$. Since we have already established that $X^* \equiv X'$, this means that also $X^*(i) \neq 0$. Thus, when applying the zero output test to X^*, by the analysis in [44, Proposition 3.9], the verifier rejects with probability $1 - \frac{O(m\cdot|\mathbb{H}|)}{|\mathbb{F}|}$. □

Thus, we have established that the proof Q^* is the canonical proof for an input $x' \in S_{A,b}$. We will now show that this input must be close to x.

Claim 4.36. *If x' is δ-far from x, then the verifier rejects with probability $\Omega(\delta)$.*

Proof. If x' is δ-far from x then with probability δ over the choice of $i \in [n]$, it holds that $x_i' \neq x_i$. Accounting for the error in the self-correction procedure, we have that the verifier rejects with probability $\Omega(\delta)$ in the Input Proximity Test. □

Thus, we may assume that x' is δ-close to x. In particular, using the fact that $\Delta\left(Q(x'),\widetilde{Q}\right) < \delta$ we obtain that

$$\max\left(\Delta\left(x,x'\right),\Delta\left(Q(x'),\widetilde{Q}\right)\right) < \delta\,.$$

Since $x' \in S_{A,b}$ we get a contradiction to the definition of δ. (Recall the definition of δ, Equation (4.6).)

This completes the proof of Lemma 4.28.

Linearity. We first establish that the canonical proof string Q is generated as a $\mathbb{GF}(2)$-linear function of the input x. We will later argue that the verifier can be expressed as a linear-system.

Consider first the polynomial $X : \mathbb{F}^{\ell} \to \mathbb{F}$. Each point in $[N]$ is clearly generated as a $\mathbb{GF}(2)$-linear combination of the input (since the circuit C has only parity gates). All other points in $\mathbb{H}^{\ell} \setminus [N]$ are identically 0. The rest of X is obtained as the unique low-degree extension, which is an $\mathbb{F}$-linear code. Since $\mathbb{F}$ has characteristic 2, the latter is also a $\mathbb{GF}(2)$-linear transformation.

The polynomial P_0 is the most tricky to handle. Indeed, from its definition it is self evident that it is a quadratic form over the field $\mathbb{F}$ (for the second term which handles booleanity). Nevertheless, we argue that it is a $\mathbb{GF}(2)$ linear function of the input x. The reason is that the mapping $\lambda \to \lambda^2$ is a $\mathbb{GF}(2)$-linear transformation.[31] The rest of the sumcheck polynomials are obtained as $\mathbb{F}$-linear combinations of $\mathcal{P}_0$, which in particular are $\mathbb{GF}(2)$-linear and so they are a $\mathbb{GF}(2)$-linear combination of the input x. Lastly, the polynomial Q is obtained by interpolation, which is a linear operation over $\mathbb{F}$ (and therefore also over $\mathbb{GF}(2)$).

As for the verifier's checks, it is easy to see that all, except the X vs. P_0 test, are $\mathbb{F}$-linear and therefore also $\mathbb{GF}(2)$-linear. The X vs. P_0 test involves an $\mathbb{F}$ quadratic form of the same form as that analyzed above, and similarly, it can be shown to be $\mathbb{GF}(2)$-linear.

4.4.2 Proof of Theorem 4.27

Fix $A \in \{0,1\}^{n \times k}$ and $b \in \{0,1\}^k$. Applying Lemma 4.28 with respect to a finite field of order $|\mathbb{F}| = O(m^2 \cdot n^{2c/m})$, we have that $S_{A,b}$ has a strongly canonical PCPP $(\mathcal{P}, \mathcal{V})$ with all the properties listed in the statement of Lemma 4.28. In particular, the proof string for an input $x \in \{0,1\}^n$ is a total degree $d = O(m \cdot n^{c/m})$-polynomial $Q : \mathbb{F}^{\ell'} \to \mathbb{F}$, where $\ell' = O(m)$. The query complexity is also $q = O(m^2 \cdot n^{c/m})$.

We construct a PCPP proof-system $(\mathcal{P}', \mathcal{V}')$ by making the following modification to $(\mathcal{P}, \mathcal{V})$. Given oracle access to an (alleged) PCPP proof string $Q : \mathbb{F}^{\ell'} \to \mathbb{F}$, the verifier runs the following tests:

1. A *robust* low-degree test on Q, with respect to degree d (see Lemma 4.12).

2. In addition, the PCPP verifier generates the PCPP queries $s_1, \dots, s_q \in \mathbb{F}^{\ell'}$ as in the PCPP of Lemma 4.28. However, rather than reading these points directly, our verifier chooses an additional random point s_0 and takes a degree q curve through $\{s_0, \dots, s_q\}$. Namely, we take a degree-q curve P that interpolates the points $\{s_0, \dots, s_q\}$ at $0, \dots, q$. The verifier reads all the points on Q that lie on the curve P. Thus, the verifier obtains the truth table of the (univariate) polynomial $Q \circ P$, which should have degree $q \cdot d = O(m^3 \cdot n^{2c/m}) \leq |\mathbb{F}|/100$. Our verifier first checks that $Q \circ P$ is indeed a degree $O(m^3 \cdot n^{2c/m})$ polynomial. It then checks that the verifier of Lemma 4.28 accepts when its answer to each query s_i is $Q \circ P(i)$, for all $i \in [q]$. If all of its checks pass then the verifier accepts.

In addition, recall that $\mathcal{V}$ only reads a single point from the input x. In order to ensure robustness, we weight this test as well as the two foregoing tests so that each test consists of one third of the queries. That is, to read a single point from the input x, we actually query the point multiple times until the number of queries to the point is one-third the total number of queries, and check that all answers are the same.

[31] To show $\mathbb{GF}(2)$ linearity it suffices to show that $(\lambda_1 + \lambda_2)^2 = \lambda_1^2 + \lambda_2^2$, which is true in $\mathbb{GF}(2)$ (also notice that multiplication by a scalar is a trivial operation in $\mathbb{GF}(2)$).

Remark 4.37. Actually, since we are aiming for a PCPP over the binary alphabet, we further "concatenate" the PCPP proof-string with a good binary linear error correcting code C. Namely, each of the $\mathbb{F}^{\ell'}$ field elements in the proof is encoded using C. The verifier is further modified so that when it is supposed to read a field element it reads the entire codeword of C, checks that it is a valid codeword (and otherwise rejects) and decodes to the corresponding symbol of $\mathbb{F}$. For simplicity, and since it only affects our parameters by a constant factor, we ignore this concatenation in the analysis below.

Robust soundness. To establish robustness we first need to show that accepting views of the PCPP verifier $\mathcal{V}'$ are at least 0.1-far apart. This follows immediately from the fact that in the two tests that the verifier preforms, it checks that the answers that it gets are low-degree polynomials (of degree d for the low-degree test and $d \cdot q$ for the additional test). Since $|\mathbb{F}| > 100d \cdot q$, there is 0.99 distance between accepting views.

We still need to establish the robust soundness condition. Namely, that for every $x \in \{0,1\}^n$ and every proof oracle Q, when $\mathcal{V}'$ is given access to the input oracle x and proof oracle Q it holds that

$$\Pr_{(I,D) \leftarrow \mathcal{V}'} \Big[\Delta\big((x \circ Q)|_I, D\big) > \rho \Big] \geq \Omega(\mu),$$

where:

$$\mu \stackrel{\text{def}}{=} \min_{x' \in S} \left\{ \max \Big(\Delta\left(x, x'\right), \Delta\left(\mathcal{P}'(x'), Q\right) \Big) \right\}.$$

Fix an input $x \in \{0,1\}^n$ and a proof oracle $Q : \mathbb{F}^{\ell'} \to \mathbb{F}$. Consider first the case that Q is 0.1-far from having degree d. In such case, by the robustness of the low degree test, with probability at least 0.5, the view of the verifier will be at least 0.1-far from accepting (see Lemma 4.9 and the discussion immediately following it). Thus, we may assume that Q is 0.1-close to some degree d polynomial Q'.

By the strongly canonical soundness of $(\mathcal{P}, \mathcal{V})$ with probability $\Omega(\mu')$, where:

$$\mu' \stackrel{\text{def}}{=} \min_{x' \in S} \left\{ \max \Big(\Delta\left(x, x'\right), \Delta\left(\mathcal{P}(x'), \pi\right) \Big) \right\}.$$

the queries $s_1, \ldots, s_q$ are such that the values $(Q(s_i))_{i \in [q]}$, would lead the verifier $\mathcal{V}$ to reject. Assume that this is indeed the case.

On the other hand, since s_0 was chosen at random, each point on the curve P, other than $s_1, \ldots, s_q$, is *individually* a random point. Since Q is 0.1-close to Q' (which has degree d), with probability $1/2$, the function $Q \circ P$ will be 0.2-close to the degree $d \cdot q$ polynomial $Q' \circ P$. To change the view of the verifier $\mathcal{V}'$ to an accepting one, an adversary must now change at least $1 - \frac{d \cdot q}{|\mathbb{F}|} - 0.2 \geq 0.7$ of the answers to the second test.

Lastly, we observe that the test that reads the input bit (required for $\mathcal{V}$) has robustness 1.

Overall we got that with probability $\Omega(\mu)$, the answers that the verifier $\mathcal{V}'$ sees are 0.1-far from any that would make it accept. The theorem follows.

Self Correction. Suppose that the PCPP string is 0.1-close to a degree $O(m \cdot n^{c/m})$ polynomial. Self correction follows from the fact that low-degree polynomials are *robust* locally correctable (see the furthermore part of Lemma 4.9).

Length of proof string. The proof string is the same as that in Lemma 4.28.

Randomness complexity. The verifier only selects one additional (random) point in $\mathbb{F}^{\ell'}$ to generate the degree-q curve P, and so has the same asymptotic randomness complexity as the verifier in Lemma 4.28.

4.5 Putting it together

To prove Theorem 4.1, we will use our composition theorem (Theorem 4.13) to perform two iterative compositions of a robust RLCC with a robust, self-correctable, strongly canonical PCPP. Specifically, we proceed in the following steps.

1. **Base Code:** Our starting point is the low-degree extension (LDE) code (aka, the Reed–Muller code), as defined in Section 4.1.3, of roughly logarithmic order. With a suitable setting of parameters, the LDE is known to be a robust (full-fledged) LCC with nearly linear block length and polylogarithmic query complexity.

2. **First Composition:** We compose this low-degree extension code with the polynomial length, polylogarithmic query, strongly canonical, self-correctable and robust PCPP that was constructed in Theorem 4.27. This composition yields a robust RLCC with polynomial block length and sublogarithmic query complexity; we denote the composed code by C'.

3. **Second Composition:** Finally, we compose the code C' with the exponential length, constant query, strongly canonical, self-correctable PCPP from Theorem 4.26. This yields our final code: an RLCC with polynomial block length and constant query complexity.

In what follows, we provide the full details of the steps of the proof outlined above. Recall that Theorem 4.13 allows us to compose a robust RLCC with a robust strongly canonical PCP of proximity to obtain a robust RLCC with improved query complexity, with a relatively mild overhead to the block length.

We proceed to describe our base code, then the first composition (using Theorem 4.13) with the PCPP in Theorem 4.27, and finally the second composition (also using Theorem 4.13) with the PCPP in Theorem 4.26.

Our base code. Fix $\mathbb{F}$ to be a finite field of characteristic 2 let $\mathbb{H} \subseteq \mathbb{F}$, and let m be a dimension such that $|\mathbb{H}| = (\log(k))^c$, $m = \frac{\log(k)}{c \cdot \log\log(k)}$ and $|\mathbb{F}| = \Theta(|\mathbb{H}| \cdot m)$, where $c \geq 1$ is chosen as a large constant. Denote $n \stackrel{\text{def}}{=} |\mathbb{F}^m|$, and note that $|H|^m = k$ and that $n = |\mathbb{F}|^m = k \cdot O(m)^m \leq k^{1+1/c}$. Consider the low-degree extension code $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m} : \mathbb{F}^k \to \mathbb{F}^n$ as defined in Section 4.1.3.

$\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ is an $\mathbb{F}$-linear code with relative distance $1 - \frac{|\mathbb{H}| \cdot m}{|\mathbb{F}|} \geq 0.9$. The furthermore clause of Lemma 4.9 and the discussion underneath it implies that it is 0.5-*robust* relaxed LCC with respect to soundness $2/3$, and decoding radius $\delta_{\mathsf{radius}} = 0.1$. Furthermore, the corrector is an $\mathbb{F}$-linear function and has randomness complexity $r_{\mathsf{code}} = \log(|\mathbb{F}|^m) = \log(n)$, and query complexity $q_{\mathsf{code}} = O(|\mathbb{H}| \cdot m) = \mathrm{polylog}(n)$.

To obtain a binary code, we concatenate $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ with a good binary linear code. Since $\mathbb{F}$ has characteristic 2, the resulting code is $\mathbb{GF}(2)$-linear. Furthermore, the resulting code has constant distance

$\delta > 0$ and is an ρ_{code}-*robust* relaxed LCC, for constant $\rho_{\mathsf{code}} > 0$, with respect to constant soundness $s_{\mathsf{code}} > 0$, and constant decoding radius $\delta_{\mathsf{radius}} > 0$. Lastly, the corrector is a $\mathbb{GF}(2)$-linear function and has randomness complexity $r_{\mathsf{code}} = \log(|\mathbb{F}|^m) = \log(n)$, and query complexity $q_{\mathsf{code}} = \mathrm{polylog}(n)$.

The first composition. Let $(\mathcal{P}, \mathcal{V})$ be the PCPP proof-system for affine relations, guaranteed by Theorem 4.27, with respect to inputs of length $\ell = 2q_{\mathsf{code}} = \mathrm{polylog}(n)$ and with dimension $m = \log(\ell)/\log\log(\ell)$. This PCPP is a self-correctable strongly canonical and ρ_{pcp}-robust PCPP $(\mathcal{P}, \mathcal{V})$ (over the binary alphabet), where $\rho_{\mathsf{pcp}} = \Omega(1)$, with respect to soundness $s_{\mathsf{pcp}} = \Omega(1)$, which is a $\mathbb{GF}(2)$-linear PCPP oracle with a $\mathbb{GF}(2)$-linear verifier and self-corrector, and has query complexity $q_{\mathsf{pcp}} = \mathrm{polylog}(\ell) = \mathrm{poly}(\log\log(n))$ and randomness complexity $r_{\mathsf{pcp}} = \mathrm{polylog}(\ell) = \mathrm{poly}(\log\log(n))$.

We compose the code from step 1 (i. e., $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ concatenated with a good binary linear error correcting code) with the PCPP $(\mathcal{P}, \mathcal{V})$ using Theorem 4.13. This yields a $\mathbb{GF}(2)$-linear ρ'-robust relaxed LCC $C' : \{0,1\}^k \to \{0,1\}^{n'}$, with respect to soundness $s' = (s_{\mathsf{pcp}} \cdot s_{\mathsf{code}} \cdot {\rho_{\mathsf{code}}}^2)/4 = \Omega(1)$ and where $\rho' = \rho_{\mathsf{pcp}}/5 = \Omega(1)$, with a $\mathbb{GF}(2)$-linear corrector, block length $n' = 2n \cdot 2^{r_{\mathsf{code}}} \cdot 2^{r_{\mathsf{pcp}}} \cdot q_{\mathsf{pcp}} = \widetilde{O}(n^2) = \widetilde{O}(k^{2+2/c})$, relative distance $\delta' \geq \delta/2$, decoding radius $\delta'_{\mathsf{radius}} = \delta_{\mathsf{radius}}/4$, randomness complexity

$$r' = 2r_{\mathsf{code}} + O\big(r_{\mathsf{pcp}} + \log(q_{\mathsf{pcp}}) + \log(q_{\mathsf{code}})\big) = 2\log(n) + \mathrm{poly}(\log\log(n)),$$

and query complexity $q' = O\big(q_{\mathsf{pcp}}\big) = O(\log\log(n))$.

The second composition. Let $(\mathcal{P}', \mathcal{V}')$ be the PCPP proof-system guaranteed by Theorem 4.26 with respect to an input of length $\mathrm{poly}(\log\log(n))$. This PCPP is a self-correctable, ${\rho_{\mathsf{pcp}}}'$-robust, strongly canonical PCPP, with respect to soundness ${s_{\mathsf{pcp}}}' = \Omega(1)$, with ${q_{\mathsf{pcp}}}' = O(1)$ queries, soundness ${\rho_{\mathsf{pcp}}}' = \Omega(1)$, and randomness complexity ${r_{\mathsf{pcp}}}' = \mathrm{poly}(\log\log(n))$. The PCPP proof string is a $\mathbb{GF}(2)$-linear function of the input and the verifier and self-correctors are $\mathbb{GF}(2)$-linear functions of their answers.

We compose the code C' with the PCPP $(\mathcal{P},' \mathcal{V}')$ using Theorem 4.13. This yields a $\mathbb{GF}(2)$-linear relaxed LCC $C'' : \{0,1\}^k \to \{0,1\}^{n''}$, with respect to soundness $s'' = \Omega({s_{\mathsf{pcp}}}' \cdot s' \cdot \rho'^2) = \Omega(1)$ The composed code C'' has block length $n'' = 2n' \cdot 2^{r'} \cdot 2^{{r_{\mathsf{pcp}}}'} \cdot {q_{\mathsf{pcp}}}' = \widetilde{O}(k^{2+2/c}) \cdot \widetilde{O}(n^2) \cdot n^{o(1)} = k^{4+4/c+o(1)}$, relative distance $\delta'' \geq \delta'/2 = \Omega(1)$, decoding radius $\delta''_{\mathsf{radius}} = \delta'_{\mathsf{radius}}/4 = \Omega(1)$, and query complexity $q'' = O\big({q_{\mathsf{pcp}}}'\big) = O(1)$.

Theorem 4.1 follows by setting $c \gg 4/\varepsilon$ so that $k^{4+4/c+o(1)} = k^{4+\varepsilon}$, for the desired parameter ε from the theorem statement.

5 Constant-rate RLCC

In this section, we construct a relaxed locally correctable code (RLCC) with quasi-polylogarithmic query complexity (specifically, $\exp(O(\log\log n)^2)$) and *constant* rate (which approaches 1). Our construction is based on the recent breakthrough result of Kopparty *et al.* [37], who give constructions of constant-rate *locally testable codes* (LTC) with quasi-polylogarithmic query complexity and constant-rate full-fledged (i. e., non-relaxed) *locally correctable codes* with $\exp(O(\sqrt{\log(n) \cdot \log\log(n)}))$ query complexity.

We show that the [37] LTC is simultaneously relaxed locally correctable with quasi-polylogarithmic query complexity.

Theorem 5.1 (Constant-Rate Binary RLCC with Quasipolylogarithmic Query Complexity)**.** *For every constant $r \in (0,1)$, there exist constants $\varepsilon > 0$, $\delta > 0$, and $\delta_{\mathsf{radius}} > 0$, as well as an (explicit) infinite family of binary, linear codes $\{C_n\}_n$ with the following properties:*

- *C_n has block length n, rate r, and relative distance δ.*
- *C_n is relaxed locally correctable with query complexity* $\exp(O(\log\log n)^2)$ *and correcting radius* δ_{radius}.

and δ furthermore satisfies that

$$\delta = \max_{r<R<1} \left\{ (1-R-\varepsilon) \cdot H^{-1}\left(1-\frac{r}{R}\right) \right\},$$

where H is the binary entropy function, and H^{-1} is its inverse in the range $(0,1/2)$.

The rate–distance tradeoff for the codes in Theorem 5.1 approaches the Zyablov bound (see Footnote 5), like the codes in [37].

On the way to proving Theorem 5.1, we will construct relaxed locally correctable codes with a larger alphabet that approach the Singleton Bound, again analogous to a result of Kopparty *et al.*

Theorem 5.2 (Constant Rate RLCC with Quasipolylogarithmic Query Complexity Approaching Singleton Bound)**.** *For every constant $r \in (0,1)$ and constant $\gamma > 0$, there exists a constant $\delta_{\mathsf{radius}} > 0$ and an infinite family of codes $\{C_n\}_n$ over an alphabet $\{\Sigma_n\}_n$ with the following properties:*

- *C_n has block length n, rate r, and relative distance at least $1-r-\gamma$.*
- *C_n is relaxed locally correctable with query complexity* $\exp(O(\log\log n)^2)$ *and correcting radius* δ_{radius}.
- *The alphabet Σ_n of C_n has size at most* $|\Sigma_n| \leq \exp(\mathrm{poly}(1/\gamma)))$.
- *Viewed as a function over* bits*, C_n is $\mathbb{GF}(2)$-linear.*

In the construction, following [37], we start off with a high-rate code C of only polylogarithmic block length and relative distance $\delta = 1/\mathrm{polylog}(n)$. The code C is trivially locally correctable with a polylogarithmic number of queries: given a string w and coordinate i, read all of w, decode to the nearest codeword c and output c_i.

Our goal is now to extend the block length of C with only a mild overhead to the query complexity. We do so gradually, by iteratively applying two transformations. The first transformation is code tensoring, which squares the block length (as well as the distance and the rate). We note that tensoring does not seem to preserve local correctability (nor decodability).[32] Nevertheless, our main observation is that tensoring does preserve *relaxed* local correctability.

Thus, one could try to simply take the code C and tensor it with itself roughly $\log\log(n)$ times to obtain a code with block length n. The problem with this approach, however, is that tensoring also squares

[32]Even assuming that the base corrector makes “smooth” queries, the natural local corrector for a tensor code squares the query complexity, which we cannot afford.

the distance, which would mean that the resulting code would have very poor distance. To rectify this, [37] uses an additional transformation that amplifies the distance without raising the query complexity of the LTC too much.

Specifically, Kopparty *et al.* use the Alon–Edmonds–Luby (AEL) [1] transform after each tensoring step to amplify the distance without hurting the rate too much. Furthermore, they show that the Alon–Edmonds–Luby transform preserves local testability (and correctability). In Section 5.2, we show that the Alon–Edmonds–Luby transform additionally preserves relaxed local correctability. Thus, similarly to [37], by repeated applications of tensoring and distance amplification we obtain our desired RLCC.

Remark 5.3. The iterative approach in the [37] construction (and therefore also in ours) resembles the "zig-zag approach" that has been used in the literature for a variety of purposes, including the construction of expander graphs [45], undirected connectivity in log space [43], PCPs [14], locally testable codes [40] and doubly efficient interactive proofs [44]. See also Goldreich's survey [21].

Section organization. In Section 5.1, we formally define the operation of tensoring on codes and show that tensoring preserves relaxed local correctability without increasing the query complexity too much. In Section 5.2, we give the AEL distance amplification lemma and prove that the AEL transform preserves relaxed local correctability. In Section 5.3, we provide the binary codes with small block length that we will use for the final constructions. In Section 5.4, we use the tools and codes from the previous sections to construct the codes of Theorem 5.2 and Theorem 5.1.

5.1 Analysis of tensoring

Following [37], the basic operation used to increase the block length of a code in our construction is tensoring.

Definition 5.4 (Tensor Code)**.** Given a field $\mathbb{F}$ and an $\mathbb{F}$-linear code $C : \mathbb{F}^k \to \mathbb{F}^n$, define the tensor of C with itself, $C^2 : \mathbb{F}^{k^2} \to \mathbb{F}^{n^2}$, to be the code such that messages $m \in \mathbb{F}^{k^2}$ are encoded as follows. View m as a $k \times k$ matrix over $\mathbb{F}$ and encode each of its rows using C. Then, encode each of the columns (including those generated in the first step) also using C.

It is a well-known fact that the rows and columns of each codeword $c' \in C^2$ are themselves codewords of C.

Now we are ready to state our main lemma of this section: that tensoring preserves relaxed locally correctability.

Lemma 5.5 (Tensoring Preserves Relaxed Local Correctability)**.** Let $\mathbb{F}$ be a field, and let $C \in \mathbb{F}^n$ be a linear, relaxed locally correctable code with query complexity q, rate r, relative distance δ, and correcting radius δ_{radius}. Then $C^2 \in \mathbb{F}^{n^2}$ is a linear, relaxed locally correctable code with query complexity $O(q/\delta_{\mathsf{radius}})$, rate r^2, relative distance δ^2, and correcting radius $\delta_{\mathsf{radius}}^2/2$.

Proof. Let w in $\mathbb{F}^{n^2}$ (which we view as an $n \times n$ matrix) be a (possibly) corrupt codeword, with at most $\delta_{\mathsf{radius}}^2/2$ fraction of corruptions, and let $(i, j) \in [n]^2$ be an index to correct. Consider the i^{th} row of w. Intuitively, if the fraction of corruptions on this row is small (less than δ_{radius}) than we could simply run the base corrector on this row. However, the total number of corruptions can be as large as $(\delta_{\mathsf{radius}}^2/2)\cdot n^2$, which can be larger than n, and so it may be the case that the entire i^{th} row is corrupt. Our plan is to detect this by choosing many coordinates of this row at random and checking whether they are corrupt by invoking the base corrector on the corresponding columns.

We proceed with the proof. For a matrix $w \in \mathbb{F}^{n\times n}$, and indices $i, j \in [n]$, we denote the i^{th} row of w by $\mathsf{row}_i(w) \in \mathbb{F}^n$ and the j^{th} column of w by $\mathsf{col}_j(w) \in \mathbb{F}^n$. We denote the $(i, j)^{\text{th}}$ entry of w by $w_{i,j}$.

Description of $\mathcal{M}'$. We now describe the action that $\mathcal{M}'$, the relaxed local corrector for C^2, takes when given a string $w \in \mathbb{F}^{n\times n}$ and a coordinate pair $(i, j) \in [n]\times[n]$. Recall that we use the notation $\mathcal{M}^{w'}(i)$ to denote the result of running relaxed local corrector $\mathcal{M}$ to correct coordinate i with oracle access to a corrupted codeword $w' \in \mathbb{F}^n$.

1. Repeat $O(1/\delta_{\mathsf{radius}})$ times:
   (a) Select at random $j' \in [n]$.
   (b) Run $\mathcal{M}^{\mathsf{col}_{j'}(w)}(i)$, with error bound 0.1 (see Remark 3.3). If the result is not equal to $w_{i,j'}$ (and in particular if it is $\perp$), then output $\perp$ and abort.
2. Output $\mathcal{M}^{\mathsf{row}_i(w)}(j)$.

Correctness of $\mathcal{M}'$. Fix $w \in \mathbb{F}^n$ and $i, j \in [n]$. We first consider the case that $w \in C^2$, in which case the corrector should correctly output $w_{i,j}$. Since each row and column of w is a codeword of C (see Definition 5.4), and $\mathcal{M}$ is a relaxed local corrector for C, we know that $\mathcal{M}^{\mathsf{col}_{j'}(w)}(i) = w_{i,j'}$, for every choice of $j' \in [n]$. Therefore, $\mathcal{M}'$ will never reject in Step 1b. Rather, it will reach Step 2 and output the correct value $w_{i,j}$. Hence, the completeness condition is satisfied.

It remains to show Condition 2, which says that if w is $(\delta_{\mathsf{radius}}^2/2)$-close to a codeword $c \in C^2$, then, with probability at least $1/2$, it holds that $\mathcal{M}'$ either decodes the correct symbol $c_{i,j}$ or outputs $\perp$.

We consider the following cases:

- If $\Delta(\mathsf{row}_i(w), \mathsf{row}_i(c)) < \delta_{\mathsf{radius}}$, then by the relaxed correcting property, $\mathcal{M}^{\mathsf{row}_i(w)}(j)$ will output either the correct value $c_{i,j}$ or $\perp$ with probability at least $1/2$.
- Thus, we may assume that $\Delta(\mathsf{row}_i(w), \mathsf{row}_i(c)) \geq \delta_{\mathsf{radius}}$. That is, at least δ_{radius} fraction of the coordinates in $\mathsf{row}_i(w)$ are corrupt. We will consider the set of coordinates $j' \in [n]$ such that $w_{i,j'}$ is corrupted (i. e., $w_{i,j'} \neq c_{i,j'}$) and $\mathsf{col}_{j'}(w)$ has less than δ_{radius} fraction of corruptions. The main observations are (1) this set is large (since the overall number of corruptions is small) and (2) if our choice of j' falls in this set (which happens with constant probability), then the relaxed local correction guarantees that when we run $\mathcal{M}$ on this column, we will detect a problem in this column with high probability. Namely, $\mathcal{M}$ will either output $\perp$ or the value $c_{i,j'} \neq w_{i,j'}$ in which case we reject. Details follow.

  Define the sets J^{err} and J^{obv} as follows:

$$J^{err} = \{j' \in [n] : w_{i,j'} \neq c_{i,j'}\}$$

is the set of coordinates $j' \in [n]$ where $w_{i,j'}$ is corrupt and

$$J^{obv} = \left\{ j' \in J^{err} \mid \Delta\left(\mathsf{col}_{j'}(w), \mathsf{col}_{j'}(c)\right) < \delta_{\mathsf{radius}} \right\}$$

is the subset of coordinates j' of J^{err} where the error is obvious to the corrector: since the column indexed by j' is not too corrupted, we can correct the error in $w_{i,j'}$ by running $\mathcal{M}$ on the column indexed by j'.

We know that $|J^{err}| \geq \delta_{\mathsf{radius}} \cdot n$, by assumption. Furthermore, we argue that at most half of the columns associated with these coordinates can have more than δ_{radius} fraction of errors. Since we assumed that $\mathrm{dist}(w,c) < \delta^2_{\mathsf{radius}}/2$, there can be at most $(\delta_{\mathsf{radius}}/2) \cdot n$ columns with more than $\delta_{\mathsf{radius}} \cdot n$ errors. Because there are at least $\delta_{\mathsf{radius}} \cdot n$ columns in J^{err}, we conclude that $|J^{obv}| \geq \delta_{\mathsf{radius}} \cdot n - (\delta_{\mathsf{radius}}/2) \cdot n = (\delta_{\mathsf{radius}}/2) \cdot n$. Finally, since we sampled $O(1/\delta_{\mathsf{radius}})$ coordinates j' uniformly at random, by setting the constant in the big-O notation suitably, with probability at least 0.99, in one of the iterations we sampled a coordinate $j' \in J^{obv}$. Assuming that $j' \in J^{obv}$, it follows that $\Delta\left(\mathsf{col}_{j'}(w), \mathsf{col}_{j'}(c)\right) < \delta_{\mathsf{radius}}$. So, by the relaxed local correction property, with probability at least 0.9, $\mathcal{M}$ will output either $c_{i,j'}$ or $\perp$. In both of these cases $\mathcal{M}'$ will output $\perp$ (since $w_{i,j'} \neq c_{i,j'}$). Therefore, $\mathcal{M}'$ will output $\perp$ with probability at least $0.99 \cdot 0.9 > 1/2$.

Parameters of tensor code. Let δ, r, δ_{radius}, and q denote the distance, rate, correcting radius, and query complexity of C. It is a well-known fact that the rate and distance of C^2 are r^2 and δ^2 respectively, and furthermore that C^2 is linear. As analyzed above, the correcting radius $\delta'_{\mathsf{radius}}$ of C^2 is at least $\delta^2_{\mathsf{radius}}/2$. Finally, our corrector $\mathcal{M}'$ makes at most $O(1/\delta_{\mathsf{radius}})$ calls to the corrector for C and reads an additional $O(1/\delta_{\mathsf{radius}})$ points by itself. Since the underlying corrector $\mathcal{M}$ makes at most q queries in each call, we get that $\mathcal{M}'$ makes at most $O(q/\delta_{\mathsf{radius}})$ queries in total.

□

5.2 Analysis of distance amplification

We now analyze the Alon–Edmonds–Luby (AEL) distance amplification method [1], and show that it preserves *relaxed* local correctability. This analysis is similar to the analysis given in [37], which shows that AEL distance amplification preserves the stronger property of local correctability (although it does not directly follow since we are starting off with the weaker hypothesis of relaxed correctability).

Lemma 5.6 (Alon–Edmonds–Luby distance amplification (see [37, Lemma 6]))**.** Suppose that C is a binary linear code with relative distance δ and rate r that is relaxed locally correctable with q queries and correcting radius $\delta_{\mathsf{radius}} \leq \delta$. Then, for every $0 < \delta'_{\mathsf{radius}} < \frac{1}{2}$ and $0 < \varepsilon < 1$, there exists a code C_{AEL} that is relaxed locally correctable with query complexity $q \cdot \mathrm{poly}(1/(\varepsilon \cdot \delta_{\mathsf{radius}}))$ with correcting radius $\delta'_{\mathsf{radius}}$ with the following properties:

- C_{AEL} has relative distance at least $2 \cdot \delta'_{\mathsf{radius}}$, and rate at least $r \cdot (1 - 2 \cdot \delta'_{\mathsf{radius}} - \varepsilon)$.
- The alphabet of C_{AEL} is $\{0,1\}^p$ for some $p = \mathrm{poly}(1/(\varepsilon \cdot \delta_{\mathsf{radius}}))$.
- C_{AEL} is $\mathbb{GF}(2)$-linear.

The proof of Lemma 5.6 uses "graph samplers" which are defined next. For a graph $G = (V, E)$, a vertex $v \in V$ and a subset of vertices $S \subseteq V$, we denote the set of edges from v to S as $E(v, S)$. Similarly, given two subsets of vertices $S, T \subseteq V$, we denote the set of edges between the sets S and T by $E(S, T)$.

Definition 5.7 (Sampler Graphs (c.f. [22]))**.** Let $G = (L, R, E)$ be a bipartite d-regular graph with $|L| = |R|$. We say that G is an (ε, δ)-sampler if for every subset $S \subseteq R$, for at least $1-\delta$ fraction of vertices $v \in L$ it holds that

$$-\varepsilon \leq \frac{|E(v,S)|}{d} - \frac{|S|}{|R|} \leq \varepsilon.$$

Lemma 5.8 (Explicit Samplers (see, e. g., [22, Lemma 5.3] or [37, Lemma 1]))**.** For every $\varepsilon, \delta > 0$ and every sufficiently large $n \in \mathbb{N}$ there exists a bipartite d-regular graph $G_{n,\varepsilon,\delta} = (L, R, E)$ with $|L| = |R| = n$ and $d = \mathrm{poly}\left(\frac{1}{\varepsilon \cdot \delta}\right)$ such that $G_{n,\varepsilon,\delta}$ is an (ε, δ)-sampler. Furthermore, there exists an algorithm that takes in n, ε, δ, and a vertex v of $G_{n,\varepsilon,\delta}$, and computes the list of neighbors of v in $G_{n,\varepsilon,\delta}$ in time $\mathrm{poly}\left(\frac{\log n}{\varepsilon \cdot \delta}\right)$.

Equipped with Lemma 5.8 we are now ready to describe AEL distance amplification and prove Lemma 5.6.

5.2.1 Overview of construction and blockwise errors

We start by giving an overview of the steps. The subsequent subsections will go into details on proving each step. Recall that our goal is to construct an RLCC that corrects from a $\delta'_{\mathsf{radius}}$ fraction of errors, with distance $\geq 2\delta'_{\mathsf{radius}}$.

At a high level, we will create C_{AEL} from C by going through the following three steps:

1. Transform C into a code C_{spread} that is relaxed locally correctable from $\delta'_{\mathsf{radius}}$ fraction of errors that are *well-spread* throughout different "blocks" of the codeword (the concept of blocks in a codeword will be defined below).

2. Transform C_{spread} into a code $C_{\mathsf{concentrated}}$ that is relaxed locally correctable from $\delta'_{\mathsf{radius}}$ fraction of errors that are *concentrated* in certain blocks of the codeword.

3. Transform $C_{\mathsf{concentrated}}$ into the final code C_{AEL} that is relaxed locally correctable from $\delta'_{\mathsf{radius}}$ fraction of *adversarial* errors.

We start by defining *blockwise error patterns*, which allow us to formalize what we mean by "well-spread" and "concentrated" errors above. This definition is implicit in the proof of distance amplification in [37].

Fix n and b to be integers such that b divides n. Suppose we have a code C with alphabet Σ and block length n. We partition the indices $\{1, \ldots, n\}$ into $\frac{n}{b}$ contiguous blocks of size b each, which we denote $s_1, \ldots, s_{n/b}$.

We say that $e \in \{0,1\}^n$ is a (δ, ε, b)-blockwise error pattern if there are at most δ fraction of the blocks in which e contains more than ε fraction of ones. In other words, when there are at most $\delta \cdot \frac{n}{b}$ choices of j such that the block s_j is contains more than $\varepsilon \cdot b$ ones. Note that a (δ, ε, b)-blockwise error pattern is permitted to have any number of blocks with $\leq \varepsilon$ fraction of ones, but can only have at most δ fraction of blocks with more than ε fraction of ones. The δ fraction of blocks that exceed the ε fraction are not restricted in any way; they could potentially be all ones.

Given a string $w \in \Sigma^n$ and a codeword $c \in C$, we say that w is (δ, ε, b)-blockwise close to c if there exists a codeword $c \in C$ such that the string $e \in \{0,1\}^n$ defined as

$$e_i = \begin{cases} 1 & \text{if } w_i \neq c_i \\ 0 & \text{o/w} \end{cases}$$

is a (δ, ε, b)-blockwise error pattern. We furthermore say that w is (δ, ε, b)-blockwise close to C if there is some codeword $c \in C$ such that w is (δ, ε, b)-blockwise close to c.

Finally, we say that C is relaxed locally correctable from (δ, ε, b)-blockwise errors if there exists a relaxed local corrector $\mathcal{M}$ such that $\mathcal{M}$ works correctly on strings w that are (δ, ε, b)-blockwise close to C. We now formally define relaxed local correctability with respect to blockwise errors.

Definition 5.9 (Relaxed Local Correctability from Blockwise Errors)**.** Fix integers $n, b > 0$ such that b divides n. Let $s_1, \ldots, s_{n/b}$ be contiguous blocks of indices of length b as described above. Let $C \subseteq \Sigma^n$ be an error correcting code. Let δ, ε be in $[0,1]$.

We say that $\mathcal{C}$ is relaxed locally correctable from (δ, ε, b)-blockwise errors if there exists a polynomial time algorithm $\mathcal{M}$, with oracle access to a string $w \in \Sigma^n$ and explicit access to an index $i \in [n]$, such that the following two conditions hold.

1. If $w \in \mathcal{C}$, then $\mathcal{M}^w(i) = w_i$ with probability 1.
2. Otherwise, for all strings $w \in \Sigma^n$ such that w is (δ, ε, b)-blockwise close to some codeword $c \in C$,
$$\Pr\big[\mathcal{M}^w(i) \in \{c_i, \perp\}\big] \geq 1/2,$$
where $\perp \notin \Sigma$ is a special abort symbol.

In all the definitions above, we optionally drop b from the notation when the blocking size is understood; thus, (δ, ε, b)-blockwise errors become (δ, ε)-blockwise errors.

To give some intuition about blockwise density and how it applies in our setting, there are two parameter regimes we will be interested in. When δ is small compared to ε, we have errors that are well-spread, since there are relatively few blocks that contain more than ε fraction of errors. By contrast, when δ is large and ε is small, we have many blocks that could be entirely corrupted, but relatively few corruptions outside of them. In this case, we have errors that are concentrated within a few blocks.

Now we give a more detailed overview of each of the three major steps of the proof, using the language of blockwise density:

1. In the first step, we partition each codeword $c \in C$ into blocks, then encode each block with a Reed–Solomon code with block length d and distance $2\delta'_{\mathsf{radius}} + \varepsilon$. We will show that this gives us a new code C_{spread} that is relaxed locally correctable from $(\delta_{\mathsf{radius}}, \delta'_{\mathsf{radius}} + \varepsilon/2)$-blockwise errors. This is because these kinds of blockwise errors have the errors *well-spread*, since we think of δ_{radius} as small compared to the desired correcting radius $\delta'_{\mathsf{radius}}$.
2. In the second step, we will use the sampler graphs of Lemma 5.8 to apply a "pseudorandom" permutation[33] σ to the indices of codewords of C_{spread} to transform C_{spread} into a new code

[33]That is, a specific *deterministic* permutation that satisfies some random-like properties (which will be specified below). In particular, we emphasize that we do not refer to the cryptographic primitive of the same name.

$C_{\mathsf{concentrated}}$ that is resilient to $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors. We choose the permutation σ that we apply such that the inverse permutation σ^{-1} scrambles $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors and turns them into $(\delta_{\mathsf{radius}}/2, \delta'_{\mathsf{radius}} + \varepsilon/2)$-blockwise errors. Since C_{spread} is correctable from $(\delta_{\mathsf{radius}}/2, \delta'_{\mathsf{radius}} + \varepsilon/2)$-blockwise errors,[34] this ensures that $C_{\mathsf{concentrated}}$ is relaxed locally correctable from $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors.

3. Finally, we will increase the alphabet size by collapsing blocks together into characters over a larger alphabet to create C_{AEL}. If the adversary is only permitted to corrupt $\delta'_{\mathsf{radius}}$ fraction of symbols, this limits the error patterns that the adversary can impose on the underlying code $C_{\mathsf{concentrated}}$ to $(\delta'_{\mathsf{radius}}, 0)$-blockwise error patterns. Thus, by using the corrector for $C_{\mathsf{concentrated}}$ on the resulting error patterns, we have a relaxed local corrector for C_{AEL} that works up to radius $\delta'_{\mathsf{radius}}$.

For a diagram depicting these steps, we refer the reader to Figure 1.

Remark 5.10 (Comparison to Proof of [37])**.** As previously stated, our proof that AEL distance amplification preserves relaxed local correctability is very similar to the proof that it preserves local correctability from [37]. We address the reasons for this similarity at a high level before proceeding with the proof.

Kopparty *et al.* show that AEL distance amplification constructs a bijection between the base code C and the final code C_{AEL} such that the local corrector $\mathcal{M}_{\mathsf{AEL}}$ for C_{AEL} calls the local corrector $\mathcal{M}$ for C many times as a subroutine, with the property that if all the calls to $\mathcal{M}$ were successful in correction, then $\mathcal{M}_{\mathsf{AEL}}$ will correct successfully. By sufficiently amplifying the success probability of $\mathcal{M}$ each time it is invoked, they conclude that $\mathcal{M}_{\mathsf{AEL}}$ succeeds with probability at least 2/3.

We now observe that a similar story can be told for relaxed correctors. We suppose that the code C has a relaxed local corrector $\mathcal{M}$, and we provide a relaxed local corrector $\mathcal{M}_{\mathsf{AEL}}$ for C_{AEL} that invokes $\mathcal{M}$. Just like before, whenever all the invocations to $\mathcal{M}$ correct successfully, the relaxed corrector $\mathcal{M}_{\mathsf{AEL}}$ corrects successfully. Furthermore, whenever a single invocation to $\mathcal{M}$ returns $\perp$, the corrector $\mathcal{M}_{\mathsf{AEL}}$ can deduce that there is some error in the purported codeword and can safely return $\perp$. By reducing the error probability of $\mathcal{M}$ sufficiently (see Remark 3.3), we can conclude that the probability that $\mathcal{M}_{\mathsf{AEL}}$ makes an error is less than $1/3$.

5.2.2 Setting parameters for the construction

Let C, r, δ_{radius}, $\delta'_{\mathsf{radius}}$, q, and ε be as in Lemma 5.6. Let n be the block length of C. Let $\{G_n\}_n$ denote an explicit, infinite family of $(\delta_{\mathsf{radius}}/2, \varepsilon/2)$-sampler graphs as given by Lemma 5.8, and let d be their degree. Define b to be $(1 - 2\delta'_{\mathsf{radius}} - \varepsilon) \cdot d$.

We will use the sampler graphs from Lemma 5.8 to construct the "pseudo-random" permutations used to scramble errors in Step 2. Additionally, d, the degree of the sampler graphs, must match the size of the blocks in Step 2 (see Lemma 5.12). Hence, we are forced to select b to be small enough with respect to d so that the blocks in Step 1 can tolerate $2\delta'_{\mathsf{radius}} + \varepsilon$ fraction of errors.

[34] We use $\delta_{\mathsf{radius}}/2$ as opposed to δ_{radius} here in order to prove that the distance of the resulting codes is $2\delta'_{\mathsf{radius}}$. This appears in Lemma 5.13.

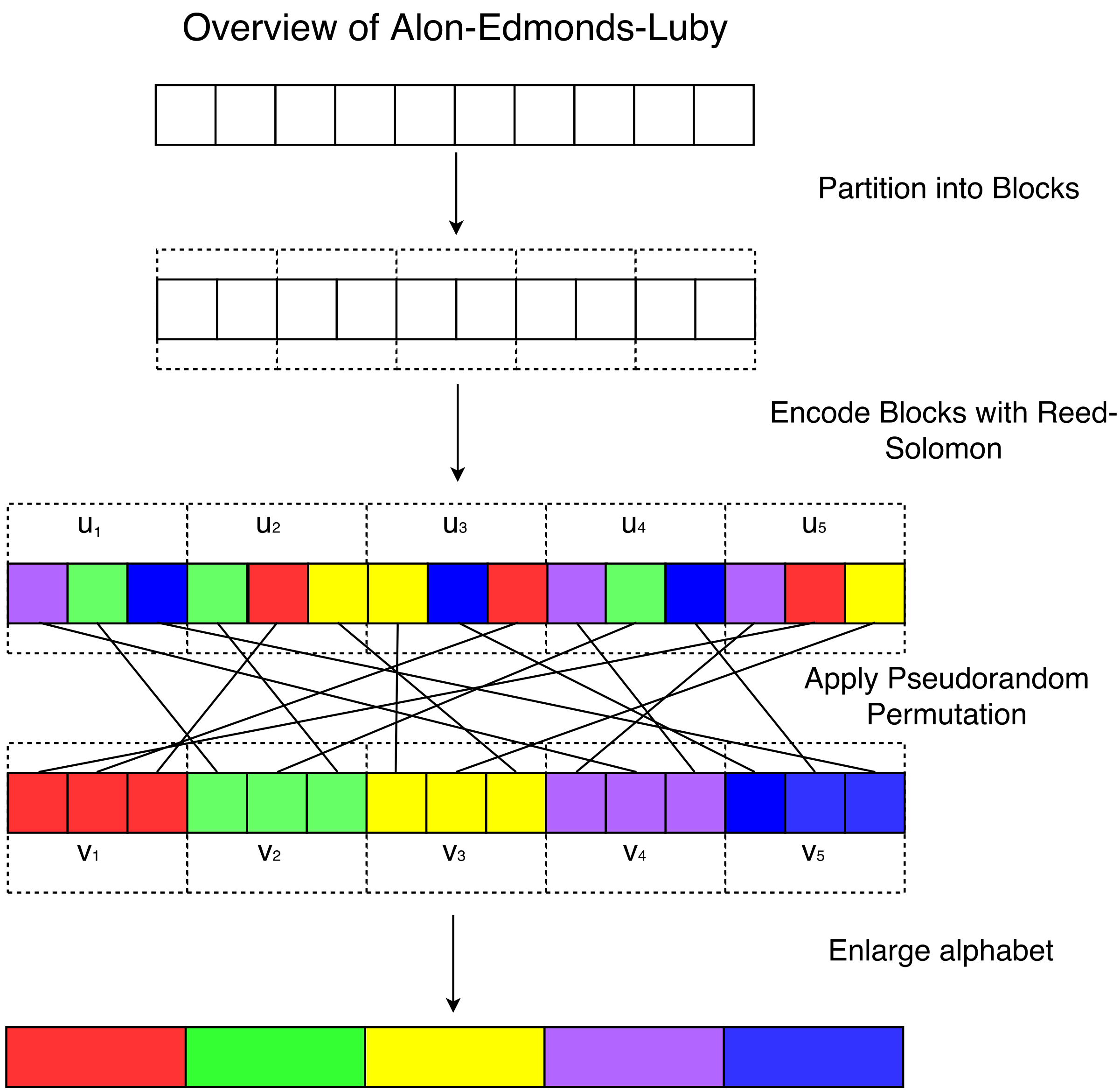

Figure 1: A high-level overview of the Alon–Edmonds–Luby transform. In the first step, a codeword is partitioned into equal-sized blocks, depicted here as length 2. Then, each block is encoded with a Reed–Solomon code. The encoded blocks are associated with vertices on one side of a bipartite graph with good sampling properties, and the permutation is applied according to the edges of the graph. The sampler is chosen such that the "concentrated" $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors in the resulting code are sufficiently scrambled after applying the inverse permutation. Finally, the alphabet is enlarged to restrict the adversary to $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors.

5.2.3 Encoding for well spread errors

We start by describing the first transformation from the code C to the code C_{spread}. Consider a codeword $c \in C$. Recall that C is a binary code, so c is a string in $\{0,1\}^n$. Let $t = \lceil \log d \rceil$.

1. Let $\mathbb{F}_{2^t}$ be a finite field of order 2^t. (Note that in particular $\mathbb{F}_{2^t}$ has characteristic 2.)

   Let $B : \mathbb{F}_{2^t}^b \to \mathbb{F}_{2^t}^d$ be the Reed–Solomon code over $\mathbb{F}$ with block length d, message length b, distance $2\delta'_{\mathsf{radius}} + \varepsilon$, and rate $1-(2\delta'_{\mathsf{radius}} + \varepsilon)$. Note that B basically has our target correcting radius, but with small block length. The rest of this first transformation will use B to modify C and endow it with some of B's error correcting properties.

2. Partition the codeword c into $\frac{n}{b\cdot t}$ blocks of length $b\cdot t$. If $b\cdot t$ does not divide the length of c, simply pad c with 0's so that it does. This adds at most $b\cdot t$ to the block length, which is negligible if the block length is at least $\frac{b\cdot t}{\varepsilon}$. If it is less than $\frac{b\cdot t}{\varepsilon}$, then a Reed–Solomon code with block length $n \le \frac{b\cdot t}{\varepsilon}$ suffices, since the block length is so small that we can afford to read the whole thing to correct. We will interpret each block as a string of length b over the alphabet $\{0,1\}^t$.

3. Encode each block using the code B. Each encoded block has length d (with alphabet $\mathbb{F}_{2^t}$), so the total length of all the encoded blocks is $\frac{n}{b\cdot t}\cdot d$.

Note that $(\delta_{\mathsf{radius}}, \delta'_{\mathsf{radius}} + \varepsilon/2)$-blockwise error patterns only have δ_{radius} fraction of blocks that have more than $\delta'_{\mathsf{radius}} + \varepsilon/2$ fraction of errors. Any block that gets at most $\delta'_{\mathsf{radius}} + \varepsilon/2$ fraction of errors can be corrected by using the Reed–Solomon corrector. The δ_{radius} fraction of blocks that are so corrupted that the Reed–Solomon corrector will err are few enough that they can be handled by the local corrector for C. We now formalize this intuition.

Lemma 5.11 (C_{spread} Relaxed Locally Correctable from "Well-Spread" Errors)**.** Let C, δ, δ_{radius}, $\delta'_{\mathsf{radius}}$, d, q, and ε be as defined in Section 5.2.2. Let B be the Reed–Solomon Code with distance $2\delta'_{\mathsf{radius}} + \varepsilon$ described in the construction of C_{spread} above.

The code $C_{\mathsf{spread}} : \{0,1\}^k \to \mathbb{F}_{2^t}^{d\cdot n/(b\cdot t)}$ (over the alphabet $\mathbb{F}_{2^t}$) is $\mathbb{GF}(2)$-linear and relaxed locally correctable from $(\delta_{\mathsf{radius}}, \delta'_{\mathsf{radius}} + \varepsilon/2)$-blockwise errors. Furthermore, for any two distinct codewords $c^{(1)}_{\mathsf{spread}}, c^{(2)}_{\mathsf{spread}}$ in C_{spread}, there exists a set of blocks of measure at least δ such that $c^{(1)}_{\mathsf{spread}}$ and $c^{(2)}_{\mathsf{spread}}$ differ in at least $(2\delta'_{\mathsf{radius}} + \varepsilon)$-fraction of coordinates in each of these blocks.

Finally, the corrector $\mathcal{M}_{\mathsf{spread}}$ for C_{spread} makes at most $q \cdot \mathrm{poly}(d)$ queries.

Proof. Note that C_{spread} is clearly $\mathbb{GF}(2)$-linear, since C was binary and linear, and using the fact that $\mathbb{F}_{2^t}$ has characteristic 2, the transformation from C to C_{spread} preserves $\mathbb{GF}(2)$-linearity.

The property that any two distinct codewords in C_{spread} differ in at least $(2\delta'_{\mathsf{radius}} + \varepsilon)$-fraction of coordinates in at least δ fraction of blocks follows from the distances of the codes C and B. Since any two codewords in C differ in at least δ fraction of coordinates, even after we partition these codewords into blocks of size b over the alphabet $\mathbb{F}_2^t$ in Step 2 of the process above, we retain the property that any two codewords transformed by this process differ in at least δ fraction of the blocks. Then, each of these blocks is treated as an input to the Reed–Solomon code B, which has distance $2\delta'_{\mathsf{radius}} + \varepsilon$, which gives us the desired claim.

Now we analyze relaxed local correctability. Suppose that our corrector $\mathcal{M}_{\mathsf{spread}}$ is given oracle access to a string $w' \in \mathbb{F}_{2^t}^{d\cdot n/(b\cdot t)}$ that is $(\delta_{\mathsf{radius}}, \delta'_{\mathsf{radius}} + \frac{\varepsilon}{2})$-blockwise close to a codeword $c_{\mathsf{spread}} \in C_{\mathsf{spread}}$, and an index $i \in [d \cdot n/(b \cdot t)]$ to correct. Let c denote the codeword in C that was transformed into c_{spread}.

Let us first pretend that $\mathcal{M}_{\mathsf{spread}}$ has oracle access to c. Fix $j \in [\frac{n}{b\cdot t}]$ such that the index i belongs to the j^{th} block. Then correcting index i is easy: $\mathcal{M}_{\mathsf{spread}}$ can query all the bits of c that lie in the j^{th} block, encode these bits using B, and read off $c_{\mathsf{spread}}[i]$ correctly from the resulting string.

It would be great if we could simulate direct oracle access to c. Instead, we do the next best thing: we show how $\mathcal{M}_{\mathsf{spread}}$ can simulate $\mathcal{M}$, the relaxed local corrector of C, on a string w that is δ_{radius}-close to c and has the property that $w' = c_{\mathsf{spread}} \Rightarrow w = c$.

Consider replacing every query to c_k with a call to $\mathcal{M}$ with string w and index k. Since $\mathcal{M}$ has perfect completeness, it is clear that we get perfect completeness for $\mathcal{M}_{\mathsf{spread}}$, since in this case, $w = c$.

Now we analyze the soundness. We note that we can amplify the success probability of $\mathcal{M}$ (Remark 3.3) whenever we call it so that the probability of error in the soundness case is at most $\frac{1}{3b\cdot t}$. This costs us a multiplicative factor of $O(\log(b \cdot t)) = \mathrm{poly}(d)$ in the query complexity of $\mathcal{M}_{\mathsf{spread}}$. In return, we get that the probability that $\mathcal{M}_{\mathsf{spread}}$ sees a response from $\mathcal{M}$ that is incorrect (i. e., not $\perp$ or consistent with c) in this whole process is at most 1/3.

If $\mathcal{M}_{\mathsf{spread}}$ ever sees $\mathcal{M}$ output $\perp$, $\mathcal{M}_{\mathsf{spread}}$ immediately aborts and outputs $\perp$, which is valid since the transformation from C to C_{spread} guarantees that $w \neq c \Rightarrow w' \neq c_{\mathsf{spread}}$. Hence, $\mathcal{M}_{\mathsf{spread}}$ will either output $c_{\mathsf{spread}}[i]$ or $\perp$ with probability at least 2/3.

Now we describe how $\mathcal{M}_{\mathsf{spread}}$ simulates $\mathcal{M}$ on w. Whenever $\mathcal{M}$ wishes to query a coordinate, $\mathcal{M}_{\mathsf{spread}}$ reads the entire *block* of w' that corresponds to the encoding of the block in which this coordinate lies. If this block is not a valid codeword of B, $\mathcal{M}_{\mathsf{spread}}$ rejects and outputs $\perp$, so wlog we will assume that all the blocks read are valid codewords of B. So, the corrector decodes this block of w', and answers the query accordingly. This process of answering a single query made by $\mathcal{M}$ costs d queries. We call the implicit string from which $\mathcal{M}$ receives responses to its queries w, which is well-defined since we assume that all the blocks that $\mathcal{M}_{\mathsf{spread}}$ queries are valid codewords.[35] Note that $w' = c_{\mathsf{spread}} \Rightarrow w = c$.

Note that, since the distance of B was $2\delta'_{\mathsf{radius}} + \varepsilon$, $\mathcal{M}_{\mathsf{spread}}$ will decode any block that has at most $\delta'_{\mathsf{radius}} + \varepsilon/2$ fraction of errors correctly.[36]

By the assumption that w is $(\delta_{\mathsf{radius}}, \delta'_{\mathsf{radius}} + \varepsilon/2)$-blockwise close to C_{spread}, at least $(1 - \delta_{\mathsf{radius}})$ fraction of the blocks of w' have at most $(\delta'_{\mathsf{radius}} + \varepsilon/2)$ fraction of errors, we know that the implicit string w, from which $\mathcal{M}_{\mathsf{spread}}$ provides answers to the queries of $\mathcal{M}$, is δ_{radius}-close to C. Hence, by assumption, $\mathcal{M}_{\mathsf{spread}}$ can simulate the relaxed local corrector $\mathcal{M}$ correctly. $\square$

5.2.4 From "well-spread" to "concentrated" errors

In this step we start with the code C_{spread} that is relaxed locally correctable from $(\delta_{\mathsf{radius}}, \delta'_{\mathsf{radius}} + \frac{\varepsilon}{2})$-blockwise errors with query complexity $q \cdot \mathrm{poly}(d)$ (see Lemma 5.11), and describe how to transform it into a code $C_{\mathsf{concentrated}}$ that is relaxed local correctable from $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors. Recall by the

[35] Of course, w is only defined on the coordinates in blocks where $\mathcal{M}_{spread}$ makes queries.

[36] As noted by a reviewer, it will actually decode any block with strictly fewer than $2\delta'_{\mathsf{radius}} + \varepsilon$ errors correctly, as fewer errors will lead to a non-codeword. Since we only desire a constant correcting radius for our final results, we sacrifice a factor of two to avoid off-by-one issues.

definition of blockwise error patterns that this is equivalent to being resilient against adversaries who can choose a $\delta'_{\mathsf{radius}}$ fraction of blocks to *fully* corrupt, but cannot corrupt any other coordinates.

The high level idea to cope with this is to apply a "pseudorandom" permutation to the coordinates of C_{spread} to get a code $C_{\mathsf{concentrated}}$. As a warmup, we will outline the argument for a random permutation $\sigma : [\frac{n}{b\cdot t}\cdot d] \to [\frac{n}{b\cdot t}\cdot d]$. Let

$$\sigma(C_{\mathsf{spread}}) = \left\{ c_{\sigma^{-1}(1)} \circ \ldots \circ c_{\sigma^{-1}\left(\frac{n}{b\cdot t}\cdot d\right)} \;:\; c \in C_{\mathsf{spread}} \right\}$$

be the set of strings that results from applying σ to the coordinates of each codeword $c_{\mathsf{spread}} \in C_{\mathsf{spread}}$. Just like for strings in C_{spread}, we will view strings $c \in \sigma(C_{\mathsf{spread}})$ as a sequence of $\frac{n}{b\cdot t}$ blocks of length d.

Now we sketch why $\sigma(C_{\mathsf{spread}})$ is a code that is relaxed locally correctable from $(\delta'_{\mathsf{radius}}, 0)$-blockwise error patterns whp. Consider a string w such that w is $(\delta'_{\mathsf{radius}}, 0)$-blockwise close to $\sigma(C_{\mathsf{spread}})$. Let c be a codeword of $\sigma(C_{\mathsf{spread}})$ that is $(\delta'_{\mathsf{radius}}, 0)$-blockwise close to w, and let c' be the codeword of C_{spread} that corresponds to c under the permutation σ, i. e., $c' = c_{\sigma(1)} \circ \ldots \circ c_{\sigma(\frac{n}{b\cdot t}\cdot)}$. Since σ was a random permutation, we can apply a Chernoff bound to conclude that $c_{\sigma(1)} \circ \ldots \circ c_{\sigma(\frac{n}{b\cdot t}\cdot d)}$ differs from w on less than $\delta'_{\mathsf{radius}} + \frac{\varepsilon}{2}$ fraction of coordinates on all but $\delta_{\mathsf{radius}}/2$ fraction of blocks, and we can use the corrector for C_{spread}.

While the fact that a random permutation works is nice, in order to make our code $C_{\mathsf{concentrated}}$ explicit we need to use an explicit permutation. Now we observe that we do not actually need σ to be a truly random permutation—a cleverly selected "pseudorandom" permutation will suffice. The property of σ used was that it *mixes* up coordinates well enough that, if we start with a $(\delta'_{\mathsf{radius}}, 0)$-blockwise error pattern e, the permuted string $e_{\sigma(1)} \circ \ldots \circ e_{\sigma\left(\frac{n}{b\cdot t}\cdot d\right)}$ has at most $\delta_{\mathsf{radius}}/2$ fraction of blocks that have more than a $\delta'_{\mathsf{radius}} + \frac{\varepsilon}{2}$ fraction of ones—that is, it is a $(\delta_{\mathsf{radius}}/2, \delta'_{\mathsf{radius}} + \frac{\varepsilon}{2})$-blockwise error pattern.

Lemma 5.12 (Explicit Permutations That Sufficiently Mix)**.** Fix n, b, d, t, δ_{radius}, ε, and $\delta'_{\mathsf{radius}}$ to be as in Section 5.2.2. There exists an (explicit) permutation $\sigma : [\frac{n}{b\cdot t}\cdot d] \to [\frac{n}{b\cdot t}\cdot d]$ such that, for every $(\delta'_{\mathsf{radius}}, 0, d)$-blockwise error pattern $\bar{e} \in \mathbb{F}_{2^t}^{\frac{n}{b\cdot t}\cdot d}$, the permuted string $\bar{e}_{\sigma(1)} \circ \ldots \circ \bar{e}_{\sigma(\frac{n}{b\cdot t}\cdot d)}$ is a $(\delta_{\mathsf{radius}}/2, \delta'_{\mathsf{radius}} + \frac{\varepsilon}{2}, d)$-blockwise error pattern.

We recall that d depends on δ_{radius} and ε as it is set using Lemma 5.8.

Proof. Towards the goal of constructing σ, we describe a correspondence between d-regular, simple, bipartite graphs $G = (L, R, E)$ where $|L| = |R| = \frac{n}{b\cdot t}$ and permutations $\pi(G)$ on $[\frac{n\cdot d}{b\cdot t}]$. Each of the $\frac{n}{b\cdot t}$ blocks of d indices will be associated with one vertex in L and one vertex in R. For each vertex, we assign an arbitrary order to its incident edges. If an edge e is the i^{th} incident edge on vertex $u \in L$ and the j^{th} incident edge on vertex $v \in R$, then we let $\pi(G)$ take the ith position in the block corresponding to u to the jth position in the block corresponding to v.

Recall the property we are looking for in the permutation σ - we want that, for all $(\delta'_{\mathsf{radius}}, 0)$-blockwise error patterns $\bar{e}$, the permuted string $\bar{e}_{\sigma(1)} \circ \ldots \circ \bar{e}_{\sigma(\frac{n}{b\cdot t}\cdot d)}$ has at most $\delta_{\mathsf{radius}}/2$ fraction of blocks that have more than $\delta'_{\mathsf{radius}} + \frac{\varepsilon}{2}$ fraction of ones. By replacing blocks with vertices and identifying the $\delta'_{\mathsf{radius}}$ fraction of corrupted blocks of $\bar{e}$ with the vertex set S in the definition of sampler graphs (Definition 5.7) and interpreting the edges of the graph as describing a mapping between indices, we can see that this property exactly corresponds to G being a $(\delta_{\mathsf{radius}}/2, \varepsilon/2)$-sampler. Hence, by taking a d-regular $(\delta_{\mathsf{radius}}/2, \varepsilon/2)$-sampler with $\frac{n}{b \cdot t}$ vertices on each side guaranteed by Lemma 5.8 and taking the corresponding permutation, we can construct sufficiently mixing permutations in time $\mathrm{poly}\left(\frac{\log(n)}{\delta_{\mathsf{radius}} \cdot \varepsilon}\right) = \mathrm{poly}(\log(n) \cdot d)$. □

Let σ be the explicit permutation given by Lemma 5.12. We define $C_{\mathsf{concentrated}} = \sigma(C_{\mathsf{spread}})$.

Lemma 5.13. Let $\delta'_{\mathsf{radius}}, q$, and d be defined as in Section 5.2.2. The code $C_{\mathsf{concentrated}}$ is $\mathbb{F}_2$-linear. Furthermore, every pair of distinct codewords $c^{(1)}_{\mathsf{concentrated}}, c^{(2)}_{\mathsf{concentrated}}$ in $C_{\mathsf{concentrated}}$ differ in at least $2\delta'_{\mathsf{radius}}$ fraction of blocks.

Finally, $C_{\mathsf{concentrated}}$ is relaxed locally correctable from $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors, where the relaxed local corrector makes at most $q \cdot \mathrm{poly}(d)$ queries.

Proof. Let σ be the permutation used in the definition of $C_{\mathsf{concentrated}}$. Let ε and δ_{radius} be as defined in Section 5.2.2. Note that applying a permutation preserves $\mathbb{GF}(2)$-linearity, so $C_{\mathsf{concentrated}}$ is $\mathbb{GF}(2)$-linear.

The property that every pair of distinct codewords in $C_{\mathsf{concentrated}}$ differ in at least $2\delta'_{\mathsf{radius}}$ fraction of blocks follows from the fact that σ maps $(\delta'_{\mathsf{radius}}, 0)$-blockwise error patterns to $(\delta_{\mathsf{radius}}/2, \delta'_{\mathsf{radius}} + \frac{\varepsilon}{2})$-error patterns. Therefore, σ must map $(2\delta'_{\mathsf{radius}}, 0)$-blockwise error patterns to $(\delta_{\mathsf{radius}}, 2\delta'_{\mathsf{radius}} + \varepsilon)$-blockwise error patterns, which follows from splitting the $(2\delta'_{\mathsf{radius}}, 0)$-blockwise error pattern into two $(\delta'_{\mathsf{radius}}, 0)$-blockwise error patterns and taking a union bound.

Consider fixing any codeword $c_{\mathsf{concentrated}} \in C_{\mathsf{concentrated}}$. We argue that any string w that is $(2\delta'_{\mathsf{radius}} - \gamma, 0)$-blockwise close to $c_{\mathsf{concentrated}}$ for any $\gamma > 0$ cannot be a codeword of $C_{\mathsf{concentrated}}$, which establishes the claim. Suppose for contradiction that w was a codeword of $C_{\mathsf{concentrated}}$, and let c_{spread} be the codeword of C_{spread} that corresponds to $c_{\mathsf{concentrated}}$. By applying the permutation σ^{-1} on the coordinates of w, we get some codeword $w_{\mathsf{spread}} \in C_{\mathsf{spread}}$ by the definition of $C_{\mathsf{concentrated}}$. However, due to the mixing property of σ described above, we must have that w_{spread} is $(\delta_{\mathsf{radius}}, 2\delta'_{\mathsf{radius}} + \varepsilon - \gamma)$-blockwise close to c_{spread}. This is a contradiction, since two distinct codewords of C_{spread} should differ by at least $2\delta'_{\mathsf{radius}} + \varepsilon$ fraction of symbols in at least $\delta \geq \delta_{\mathsf{radius}}$ fraction of blocks.

We now describe the relaxed local corrector. This corrector will simply apply the permutation σ to the coordinates of purported codewords of $C_{\mathsf{concentrated}}$ and run the relaxed local corrector of C_{spread} on the resulting string. This suffices due to the mixing property of the permutation σ^{-1}.

Fix a string z' that is $(\delta'_{\mathsf{radius}}, 0)$-blockwise close to a codeword $c' \in C_{\mathsf{concentrated}}$ and a coordinate $i \in [\frac{n}{b \cdot t} \cdot d]$ to correct. Run the relaxed local corrector for C_{spread} with input string $z'' = z'_{\sigma(1)} \circ \ldots \circ z'_{\sigma(\frac{n}{b \cdot t} \cdot d)}$ and coordinate $\sigma^{-1}(i)$. Whenever the corrector queries a position j, provide it with z''_j. Return whatever the relaxed local corrector for C_{spread} returns.

To establish soundness, note that, due to the properties of σ given by Lemma 5.12, the string z'' will have at most $\delta_{\mathsf{radius}}/2$ fraction of blocks with more than $\delta'_{\mathsf{radius}} + \frac{\varepsilon}{2}$ fraction of errors. Since we have a relaxed local corrector for C_{spread}, we will recover $z''_{\sigma^{-1}(i)} = z'_i$ or $\perp$ with high probability due to the soundness property of the relaxed local corrector for C_{spread} (see Lemma 5.11).

To establish completeness, note that when $z' = c'$, we will always return c'_i. This is because the string $z'' := c'_{\sigma(1)} \circ \ldots \circ c'_{\sigma(\frac{n}{b \cdot t} \cdot d)}$ is a valid codeword of C_{spread}, and so the corrector for C_{spread} will always correctly return $z''_{\sigma^{-1}(i)}$ by the completeness property of relaxed local correction.

The number of queries made by this corrector for $C_{\mathsf{concentrated}}$ is exactly the same as the number of queries used by the corrector for C_{spread}. □

Finally, we describe how to transform $C_{\mathsf{concentrated}}$ into a code that is resilient to *adversarial* errors.

5.2.5 From blockwise to adversarial errors

We start from a code $C_{\mathsf{concentrated}}$ that is relaxed locally correctable for $(\delta'_{\mathsf{radius}}, 0)$-blockwise error patterns as described above, and construct a code C_{AEL} that is relaxed locally correctable for adversarial errors.

We form C_{AEL} from $C_{\mathsf{concentrated}}$ by taking each codeword in $C_{\mathsf{concentrated}}$ and rewriting each block of symbols $(c_1, \ldots, c_d) \in \mathbb{F}_{2^t}^d$ into a single character in $\mathbb{F}_{2^{t \cdot d}}$ in the natural way (i. e., by viewing d-dimensional vectors over $\mathbb{F}_{2^t}$ as an element of the field $\mathbb{F}_{2^{t \cdot d}}$)

The idea is that, now, adversarial errors for C_{AEL} correspond to blockwise errors in the code $C_{\mathsf{concentrated}}$, which we can correct with the relaxed local corrector for $C_{\mathsf{concentrated}}$.

We now prove Lemma 5.6.

Proof of Lemma 5.6. The corrector for C_{AEL}, when asked to correct a coordinate i, will simply use the corrector for $C_{\mathsf{concentrated}}$ to correct each of the coordinates $(i-1) \cdot d + 1, \ldots, i \cdot d$. Since the corrector for $C_{\mathsf{concentrated}}$ has query complexity $q \cdot \mathrm{poly}(d)$, we conclude that C_{AEL} also has query complexity $q \cdot \mathrm{poly}(d) = q \cdot \mathrm{poly}\left(\frac{1}{\varepsilon \cdot \delta_{\mathsf{radius}}}\right)$. Furthermore, since $C_{\mathsf{concentrated}}$ was locally correctable from $(\delta'_{\mathsf{radius}}, 0)$-blockwise errors, we see that C_{AEL} has correcting radius $\delta'_{\mathsf{radius}}$.

We begin by proving the rate and distance of the overall AEL transform we have described.

The rate of C_{AEL} can be written as

$$r_{\mathsf{AEL}} = \frac{\log|C|}{(n/(b \cdot t)) \cdot (d \cdot t)}.$$

Noting that the rate of C is $r = \frac{\log|C|}{n}$ and simplifying, we get that

$$r_{\mathsf{AEL}} \geq r \cdot \frac{b}{d}.$$

Plugging in that $b = d(1 - 2\delta'_{\mathsf{radius}} - \varepsilon)$, we get that

$$r_{\mathsf{AEL}} \geq (1 - 2\delta'_{\mathsf{radius}} - \varepsilon) \cdot r$$

as desired.

The distance of the code C_{AEL} is at least $2\delta'_{\mathsf{radius}}$, which follows immediately from the fact that any two distinct codewords of $C_{\mathsf{concentrated}}$ differ in at least $2\delta'_{\mathsf{radius}}$ fraction of blocks (Lemma 5.13).

A relaxed local corrector for C_{AEL} works as follows. Given a string $a \in \mathbb{F}_{2^{t \cdot d}}^{n/(b \cdot t)}$ that is $\delta'_{\mathsf{radius}}$-close to C_{AEL} and desired coordinate $i \in [n/(b \cdot t)]$ for correction, expand the alphabet of a in the natural way to get a string $a' \in \mathbb{F}_{2^t}^{d \cdot n/(b \cdot t)}$ that is $(\delta'_{\mathsf{radius}}, 0, d)$-blockwise close to $C_{\mathsf{concentrated}}$. Run the relaxed local corrector for $C_{\mathsf{concentrated}}$ on a' for each of the coordinates in the i^{th} block.

Note that we can amplify each of these results so that the relaxed corrector for $C_{\mathsf{concentrated}}$ works correctly on each coordinate with probability $1 - \frac{1}{3d}$ at a $O(\log d)$ cost in query complexity, and so we will get the whole block correct with probability at least $2/3$ by a union bound.

Note that this operation of collapsing blocks trivially preserves $\mathbb{GF}(2)$-linearity, so C is $\mathbb{GF}(2)$-linear. The final alphabet size is $\{0,1\}^{t \cdot d}$, where we recall that $t \cdot d = d\lceil \log d \rceil = \mathrm{poly}(1/(\varepsilon \cdot \delta_{\mathsf{radius}}))$. $\square$

5.3 Concatenation

Recall that our goal is to start with a code with constant block length and repeatedly tensor and apply the Alon–Edmonds–Luby transform to increase the block length, while preserving relaxed local correctability and distance. However, for the tensor of a code C to be well-defined, we need C to be a linear code, while the AEL transform (Lemma 5.6) only outputs a $\mathbb{GF}(2)$-linear code. Fortunately, we can transform a $\mathbb{GF}(2)$-linear code C into a binary linear code by *concatenating* C with a binary code. See Section 2.1.1 for the definition of code concatenation.

We will concatenate the code that is output by the AEL Transform (Lemma 5.6) with a good binary code, so that we do not undo the distance amplification achieved by the AEL Transform and simultaneously do not lose too much in the rate. Fortunately, sufficiently good binary codes do exist.

For most applications of concatenation in this construction we will be dealing with codes with extremely small distance. For sufficiently small δ, we can use codes that satisfy the following rate distance trade-off.

Fact 5.14 (Zyablov Bound for Small Distance (Fact 2.4 from [37]))**.** For every sufficiently small $\delta > 0$, there exists an explicit, infinite family $\{Z_n\}_n$ of binary linear codes with relative distance δ and rate r at least

$$r \geq 1 - \sqrt[3]{\delta}.$$

For our final concatenation, we will use binary codes with the best possible rate–distance tradeoff we know of: codes that meet the Gilbert–Varshamov bound.[37]. We now state a fact about the existence of linear codes of constant block length matching the Gilbert–Varshamov bound.

Fact 5.15 (Codes Achieving Gilbert–Varshamov Bound)**.** There exist (trivially explicit) binary linear codes with *constant* block length that achieve the Gilbert–Varshamov bound: namely, for any constants $\delta, \varepsilon > 0$, there exists a binary linear code with block length depending on δ and ε, distance $\delta > 0$ and rate $1 - H(\delta) - \varepsilon$, where $H(p) = -p \cdot \log(p) - (1-p) \cdot \log(1-p)$ is the binary entropy function.

We note that both the codes described have efficient decoders that decode up to a constant radius. We will use these decoders in our construction.

Now we are ready to describe the overall construction of our RLCCs.

5.4 Putting it all together: final construction and parameters

As alluded to earlier, the high-level description of this construction is that we start with a linear code with $\mathrm{poly}\log(n)$ block length and high rate, then tensor it with itself, apply the Alon–Edmonds–Luby transform,

[37]For the *Gilbert–Varshamov bound* [20, 51], see, e. g., [32, Chap. 4.2].

concatenate with a sufficiently good binary code, and repeat. We now give the exact construction and its analysis. Let $\delta := \frac{1}{(\log n)^{36}}$. We first construct an intermediate RLCC with subconstant distance by iterating the AEL transform, concatenation, and tensoring. This part of the construction is identical to the construction of LTCs in [37], except that they combine the AEL transform and concatenation into what they call $\square$-distance amplification. We expand on this in Remark 5.17.

Lemma 5.16 (RLCC with Sub-constant Distance (Analogous to Lemma 5.2 in [37]))**.** There exists an explicit infinite family of binary linear codes $\{W_n\}_n$ satisfying:

1. W_n has block length at least n, rate at least $1 - O(1/\log(n))$, and relative distance at least $1/\mathrm{polylog}(n)$.
2. W_n is relaxed locally correctable with query complexity $(\log n)^{\log\log(n)}$, and constant correcting radius.

Proof. We give an algorithm to construct W_n given n.

1. Let D_0 be a code from Fact 5.14 with block length $\mathrm{polylog}(n)$, rate $1 - \sqrt[3]{\delta}$, and relative distance δ.
2. For $i = 0, \dots, \log\log(n) - 1$:
   (a) Let $\widetilde{D}_i$ be the result of applying Lemma 5.6 with input code D_i and target distance $\delta'_{\mathsf{radius}} = \sqrt[4]{\delta}/2$ and $\varepsilon = \sqrt[4]{\delta}$. Let Γ denote the output alphabet of $\widetilde{D}_i$.
   (b) Let Z be an explicit binary code with distance $\sqrt[4]{\delta}$ and rate at least $1 - \sqrt[12]{\delta}$ as guaranteed by Fact 5.14 where Z has sufficiently large message length to be concatenated with $\widetilde{D}_i$.[38] Let D'_i denote the concatenation of $\widetilde{D}_i$ with Z.
   (c) Set $D_{i+1} = (D'_i)^2$.
3. Return $D_{\log\log(n)}$.

Now we prove that the code $D_{\log\log(n)}$ has the required properties.

Correcting radius. We begin by observing that the correcting radius of D_i is a constant for each i. Clearly this holds for $i = 0$, as our corrector for D_0 just reads the entire codeword, so the correcting radius is constant. Assume this is true for $i - 1$, and we will show that it is true for i.

In the first step, we apply AEL distance amplification to get the distance to $\sqrt[4]{\delta}$. We note that this results from setting $\delta'_{\mathsf{radius}}$ to $\frac{\sqrt[4]{\delta}}{2}$ in Lemma 5.6, and hence the correcting radius of this code is $\frac{\sqrt[4]{\delta}}{2}$. Then, we concatenate with the code Z. Our corrector for the concatenated code D'_i will use the corrector for $\widetilde{D}_i$ and answer queries made by the corrector for $\widetilde{D}_i$ by reading the entire codeword of Z used to encode symbols of Γ, and using the underlying corrector, which corrects to a constant distance. Overall, we get that the code is correctable to a constant radius.

[38] When Z has a larger message size than the alphabet size of $\widetilde{D}_i$, we pad the alphabet to make the sizes equal. As it turns out, the code family from Fact 5.14 ensures that this incurs a negligible cost for our purposes.

Distance. We will prove by induction that the distance of D_i is δ for all i. Assume this is true for D_i, and now we want to prove that it is true for D_{i+1}. Note that, from Section 2.1.1, we know that the distance of D'_i, the concatenated code, is equal to the distance of $\widetilde{D}_i$ multiplied by the distance of Z. Therefore, we have that D'_i has distance $\sqrt{\delta}$. Since $D_{i+1} = (D'_i)^2$, and tensoring squares the distance, the claim follows.

Rate. Let us analyze how the rate of D_{i+1} differs from the rate of D_i. Denote the rate of D_i as r_i. First, we apply Lemma 5.6 with target distance $\sqrt[4]{\delta}$ and a parameter ε, which we set to be $\sqrt[4]{\delta}$. This gives us rate $r_i \cdot (1 - 2\sqrt[4]{\delta})$. Then, we concatenate with the code Z, which has rate at least $1 - \sqrt[12]{\delta}$, which gives us

$$\begin{aligned} r_{i+1} &\geq r_i \cdot (1 - 2\sqrt[4]{\delta})(1 - \sqrt[12]{\delta}) \\ &\geq r_i \cdot (1 - 3\sqrt[12]{\delta}). \end{aligned}$$

Finally, we tensor the code. Since tensoring squares the rate, this gives us that

$$r_{i+1} \geq r_i^2 \cdot (1 - 3\sqrt[12]{\delta})^2 \geq r_i^2(1 - 6\sqrt[12]{\delta}).$$

Note that we start with D_0 which has rate $1 - \sqrt[3]{\delta} > 1 - 6\sqrt[12]{\delta}$. We furthermore prove by induction that the rate of D_i is at least

$$r_i \geq (1 - 6\sqrt[12]{\delta})^{3^i}.$$

Indeed, assume the fact is true for $i - 1$. Then we have that

$$\begin{aligned} r_i &\geq r_{i-1}^2 \cdot (1 - 6\sqrt[12]{\delta}) \\ &\geq (1 - 6\sqrt[12]{\delta})^{3^{i-1}\cdot 2} \cdot (1 - 6\sqrt[12]{\delta}) \\ &\geq (1 - 6\sqrt[12]{\delta})^{3^i}. \end{aligned}$$

Applying the fact that $\delta := \frac{1}{(\log n)^{36}}$ we can find the final rate:

$$\begin{aligned} r_{\log\log(n)} &\geq \left(1 - \frac{6}{\log^3 n}\right)^{3^{\log\log(n)}} \\ &\geq \left(1 - \frac{6}{\log^3 n}\right)^{\log^2 n} \\ &\geq 1 - \frac{6}{\log^3 n} \cdot \log^2 n \\ &\geq 1 - \frac{6}{\log n}. \end{aligned}$$

Queries. Our goal for this section will be to prove that each iteration of the loop contributes a multiplicative factor of $\operatorname{polylog}(n)$ queries, hence our overall number of queries is $(\log n)^{O(\log\log(n))}$. Note that D_0 is relaxed locally correctable with $\operatorname{polylog}(n)$ queries and furthermore that D_0 has constant correcting radius (since its relaxed local corrector just reads everything). We will prove by induction that, for every i from 0 to $\log\log(n)$, the number of queries needed to relaxed locally correct D_i is at most $(\log n)^{k(i+1)}$, for some constant k that we will define later. We have already established that this holds for $i = 0$. Assume below that this holds for $i-1$, and we want to prove it for i.

First, we amplify the distance of our code to $\sqrt[4]{\delta}$ using AEL distance amplification. We maintain the invariant that the distance of our code at the beginning of each loop is δ and the correcting radius is constant, Lemma 5.6 tells us that the query complexity of this new code is $(\log n)^{k(i)} \cdot (\log n)^{O(1)}$. Then, we concatenate the code with the code Z. Since we can take Z to have block length $(\log(n))^{O(1)}$, we can simulate the old corrector by reading entire codewords whenever the old corrector queried a symbol while incurring at most a polylogarithmic factor in query complexity. Finally, we tensor the code, which costs us an $O(\frac{1}{\delta}) = (\log(n))^{O(1)}$ factor by Lemma 5.5. So overall, we get that the query complexity of the relaxed local corrector for D_i is at most

$$(\log n)^{k(i)} \cdot (\log(n))^{O(1)} \cdot (\log(n))^{O(1)} \cdot (\log(n))^{O(1)} = (\log n)^{k \cdot i + O(1)}.$$

Defining k to be the $O(1)$ additive term in the exponent gives us the desired result.

□

Now we will use the intermediate RLCCs from Lemma 5.16 in order to construct RLCCs with constant rate, constant distance, and quasipolylogarithmic query complexity.

Proof of Theorem 5.2. To construct these codes, we will apply the AEL Transform to the codes of Lemma 5.16 to amplify the distance to a constant. However, doing so entirely at once will cause our alphabet size to become superconstant, as we started with subconstant distance (see Lemma 5.6 for details). We will insert a step of concatenation to rectify this issue.

1. Start with a code D of block length at least n from Lemma 5.16. Apply AEL distance amplification (Lemma 5.6) to D with target distance $\delta_{\widetilde{C}} := \gamma/10$, where γ is from the statement of Theorem 5.2, and the parameter ε in Lemma 5.6 set to $\gamma/10$. Call the resulting code $\widetilde{C}$.

2. Let Z' be an explicit binary code satisfying the conditions in Fact 5.14 with distance $(\gamma)^3/1000$, rate $1 - \frac{\gamma}{10}$, and some polylogarithmic block length appropriate for concatenation with $\widetilde{C}$. Let C' be the code that results from concatenating $\widetilde{C}$ with Z'.

3. Apply AEL distance amplification (Lemma 5.6) to C' with target distance $1 - r - \gamma$ and ε in Lemma 5.6 set to $\gamma/10$. Call the resulting code C.

4. Return C.

Distance. This follows from the fact that we used distance amplification in the final step.

Rate. Recall that Lemma 5.16 gave us codes with rate $1 - O\left(\frac{1}{\log n}\right)$. After applying AEL, we will get rate at least $\left(1 - O\left(\frac{1}{\log n}\right)\right) \cdot (1 - \gamma/5) > 1 - \frac{2\gamma}{5}$. After concatenation, Subsection 2.1.1 tells us that we will get rate at least $\left(1 - \frac{2\gamma}{5}\right) \cdot \left(1 - \frac{\gamma}{10}\right) > 1 - \frac{3\gamma}{5}$. After the final distance amplification, we will get rate at least $\left(1 - \frac{3\gamma}{5}\right) \cdot (r + \gamma - \gamma/10) \geq r$.

Queries. Recall that Lemma 5.16 gave us codes with query complexity at most $q = (\log n)^{O(\log\log n)}$ that are relaxed locally correctable from $\frac{1}{\operatorname{poly}\log n}$ fraction of errors. After applying the first AEL transform with target distance $\gamma/10$, the resulting code will be an RLCC with $q \cdot \operatorname{poly}\log(n)$ queries by Lemma 5.6. Concatenating with the code Z' will increase our queries by another multiplicative $\operatorname{poly}\log n$ factor, and then our final AEL transform will increase our queries by a multiplicative constant. Overall, we retain query complexity $(\log n)^{O(\log\log n)}$ in our final code.

Alphabet. The output alphabet of the final step is $\{0,1\}^p$, where $p = \operatorname{poly}(1/\gamma)$ by Lemma 5.6. □

Finally, we can prove Theorem 5.1 by starting with the codes from Theorem 5.2, and concatenating them with binary codes given to us by Fact 5.14.

Proof of Theorem 5.1. Start with codes from Theorem 5.2 that have rate R and distance at least $1-R-\gamma$, for a small constant $\gamma > 0$. Then, simply concatenate them with binary codes of appropriate constant block length from Fact 5.15 that have distance δ' and rate $1-H(\delta')-\gamma$. The linearity of these codes follows from the fact that the codes in Theorem 5.2 are $\mathbb{GF}(2)$-linear, and we are concatenating with a linear binary code. The query complexity is easily seen to remain quasi-polylogarithmic.

We now calculate the rate and distance of the resulting codes. The rate of these concatenated codes is $r = R \cdot (1-H(\delta')-\gamma)$. Rewriting the equation to isolate δ', we get

$$\delta' = H^{-1}\left(1 - \frac{r}{R} - \gamma\right).$$

The distance of the resulting code is $\delta = (1-R-\gamma)\cdot\delta'$. Plugging in for δ', we get

$$\delta = (1-R-\gamma)\cdot H^{-1}\left(1 - \frac{r}{R} - \gamma\right)$$

where r is the rate of our desired code and R is the rate of the code from Theorem 5.2. Noting that Theorem 5.2 allowed us to construct codes for *any* constant $R \in (0,1)$ and that H^{-1} is continuous, we can maximize the above expression over R to achieve

$$\delta = \max_{r<R<1}\left\{(1-R-\varepsilon)\cdot H^{-1}\left(1-\frac{r}{R}\right)\right\}$$

for arbitrarily small $\varepsilon > 0$. □

Remark 5.17. We used the presentation of first applying Alon–Edmonds–Luby then tensoring at each iteration in order to construct the same code that is constructed in Kopparty *et al.* [37]. Readers who are already familiar with this construction may recognize that our analysis of relaxed local correctability for the code with subconstant distance (Lemma 5.16) is simpler than the analysis of the analogous theorem for local testability in Kopparty *et al.* This is because our proof that tensoring preserves relaxed local correctability does not require any additional assumptions on the base code.

In contrast, it is not necessarily true that the tensor of a locally testable code is locally testable [50]. However, it *is* true if the base code C is square of some code $\mathcal{D}$ [52]. Therefore, Kopparty *et al.* prove a distance amplification lemma that takes in a code $C = \mathcal{D}^2$ and outputs a code $\mathcal{C}' = (\mathcal{D}')^2$, by applying the AEL Transform on the base code. Furthermore, they need to prove that this modified distance amplification method still preserves local testability.

Since we do not need to use this syntactically modified distance amplification method to prove relaxed local correctability, we use the vanilla AEL Transform to construct the same code that they construct in [37]. Since the construction is exactly the same as the one in [37], it follows that the LTCs constructed in [37] are simultaneously locally testable and relaxed locally correctable (proved here), both with quasi-polylogarithmic query complexity.

Acknowledgments

We thank Oded Goldreich for initiating a discussion that led to this paper, for insightful conversations and for encouragement. The second author would like to thank Dana Moshkovitz for useful discussions surrounding this work. Finally, we would like to thank our reviewers, who gave excellent and thorough feedback that greatly improved the readability of this manuscript.

References

[1] NOGA ALON AND MICHAEL LUBY: A linear time erasure-resilient code with nearly optimal recovery. *IEEE Trans. Inform. Theory*, 42(6):1732–1736, 1996. Preliminary version by Noga Alon, Jeff Edmonds, and Michael Luby in FOCS'95. [doi:10.1109/18.556669] 5, 45, 47

[2] SANJEEV ARORA, CARSTEN LUND, RAJEEV MOTWANI, MADHU SUDAN, AND MARIO SZEGEDY: Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. Preliminary version in FOCS'92. [doi:10.1145/278298.278306] 9, 28

[3] SANJEEV ARORA AND SHMUEL SAFRA: Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998. Preliminary version in FOCS'92. [doi:10.1145/273865.273901] 8

[4] VIKRAMAN ARVIND, PUSHKAR S. JOGLEKAR, PARTHA MUKHOPADHYAY, AND S. RAJA: Randomized polynomial-time identity testing for noncommutative circuits. *Theory of Computing*, 15(7):1–36, 2019. Preliminary version in STOC'17. [doi:10.4086/toc.2019.v015a007] 19

[5] LÁSZLÓ BABAI, LANCE FORTNOW, LEONID A. LEVIN, AND MARIO SZEGEDY: Checking computations in polylogarithmic time. In *Proc. 23rd STOC*, pp. 21–31. ACM Press, 1991. [doi:10.1145/103418.103428] 7, 15

[6] ALEXANDER BARG, ITZHAK TAMO, AND SERGE VLĂDUŢ: Locally recoverable codes on algebraic curves. *IEEE Trans. Inform. Theory*, 63(8):4928–4939, 2017. Preliminary version in ISIT'15. [doi:10.1109/TIT.2017.2700859, arXiv:1603.08876] 11

[7] ELI BEN-SASSON, ODED GOLDREICH, PRAHLADH HARSHA, MADHU SUDAN, AND SALIL P. VADHAN: Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Com-

put., 36(4):889–974, 2006. Preliminary version in STOC'04. [doi:10.1137/S0097539705446810, ECCC:TR04-021] 3, 5, 6, 7, 8, 13, 14, 15, 17

[8] ELI BEN-SASSON, PRAHLADH HARSHA, ODED LACHISH, AND ARIE MATSLIAH: Sound 3-query PCPPs are long. *ACM Trans. Comput. Theory*, 1(2):7:1–7:49, 2009. Preliminary version in ICALP'08. [doi:10.1145/1595391.1595394, ECCC:TR07-127] 6

[9] MANUEL BLUM, MICHAEL LUBY, AND RONITT RUBINFELD: Self-testing/correcting with applications to numerical problems. *J. Comput. System Sci.*, 47(3):549–595, 1993. Preliminary version in STOC'90. [doi:10.1016/0022-0000(93)90044-W] 30

[10] VIVECK R. CADAMBE AND ARYA MAZUMDAR: Bounds on the size of locally recoverable codes. *IEEE Trans. Inform. Theory*, 61(11):5787–5794, 2015. Preliminary version in NetCod'13. [doi:10.1109/TIT.2015.2477406] 11

[11] CLÉMENT L. CANONNE AND TOM GUR: An adaptivity hierarchy theorem for property testing. *Comput. Complexity*, 27(4):671–716, 2018. Preliminary version in CCC'17. [doi:10.1007/s00037-018-0168-4, arXiv:1702.05678] 12

[12] ALESSANDRO CHIESA, TOM GUR, AND IGOR SHINKAR: Relaxed locally correctable codes with nearly-linear block length and constant query complexity. In *Proc. 31st Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'20)*, pp. 1395–1411. ACM Press, 2020. [doi:10.1137/1.9781611975994.84, ECCC:TR20-113] 4, 11

[13] MARCEL DALL'AGNOL, TOM GUR, AND ODED LACHISH: A structural theorem for local algorithms with applications to coding, testing, and privacy. *Electron. Colloq. on Comput. Complexity (ECCC)*, TR20-154:1–55, 2020. [ECCC] 11

[14] IRIT DINUR: The PCP theorem by gap amplification. *J. ACM*, 54(3):12–55, 2007. Preliminary version in STOC'06. [doi:10.1145/1236457.1236459, ECCC:TR05-046] 45

[15] IRIT DINUR AND PRAHLADH HARSHA: Composition of low-error 2-query PCPs using decodable PCPs. In ODED GOLDREICH, editor, *Property Testing: Current Research and Surveys*, volume 6390 of *LNCS*, pp. 280–288. Springer, 2010. Preliminary version in FOCS'09. [doi:10.1007/978-3-642-16367-8_21] 11

[16] IRIT DINUR AND OMER REINGOLD: Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006. Preliminary version in FOCS'04. [doi:10.1137/S0097539705446962] 4, 5, 15

[17] KLIM EFREMENKO: 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012. Preliminary version in STOC'09. [doi:10.1137/090772721, ECCC:TR08-069] 3, 4

[18] KATALIN FRIEDL AND MADHU SUDAN: Some improvements to total degree tests. In *4th Israel Symp. on Theory of Computing and Systems (ISTCS'95)*, pp. 190–198. IEEE Comp. Soc., 1995. [doi:10.1109/ISTCS.1995.377032, arXiv:1307.3975] 19, 20

[19] PETER GEMMELL AND MADHU SUDAN: Highly resilient correctors for polynomials. *Inform. Process. Lett.*, 43(4):169–174, 1992. [doi:10.1016/0020-0190(92)90195-2] 19

[20] EDGAR N. GILBERT: A comparison of signalling alphabets. *Bell Sys. Tech. J.*, 31(3):504–522, 1952. [doi:10.1002/j.1538-7305.1952.tb01393.x] 57

[21] ODED GOLDREICH: Bravely, moderately: A common theme in four recent works. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pp. 373–389. Springer, 2011. [doi:10.1007/978-3-642-22670-0_26] 45

[22] ODED GOLDREICH: A sample of samplers: A computational perspective on sampling. In ODED GOLDREICH, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pp. 302–332. Springer, 2011. [doi:10.1007/978-3-642-22670-0_24] 48

[23] ODED GOLDREICH: *Introduction to Property Testing*. Cambridge Univ. Press, 2017. [doi:10.1017/9781108135252] 30

[24] ODED GOLDREICH AND TOM GUR: Universal locally verifiable codes and 3-round interactive proofs of proximity for CSP. *Electron. Colloq. on Comput. Complexity (ECCC)*, 23:1–20, 2016. [ECCC:TR16-192] 6

[25] ODED GOLDREICH AND TOM GUR: Universal locally testable codes. *Chicago J. Theoret. Comp. Sci.*, 2018(3):1–21, 2018. [doi:10.4086/cjtcs.2018.003, ECCC:TR16-042] 6, 10, 11, 12

[26] ODED GOLDREICH, TOM GUR, AND ILAN KOMARGODSKI: Strong locally testable codes with relaxed local decoders. *ACM Trans. Comput. Theory*, 11(3):17:1–17:38, 2019. Preliminary version in CCC'15. [doi:10.1145/3319907, ECCC:TR14-025] 6

[27] ODED GOLDREICH AND MADHU SUDAN: Locally testable codes and PCPs of almost-linear length. *J. ACM*, 53(4):558–655, 2006. Preliminary version in FOCS'02. [doi:10.1145/1162349.1162351, ECCC:TR02-050] 6, 9, 14, 16, 32

[28] PARIKSHIT GOPALAN, CHENG HUANG, HUSEYIN SIMITCI, AND SERGEY YEKHANIN: On the locality of codeword symbols. *IEEE Trans. Inform. Theory*, 58(11):6925–6934, 2012. [doi:10.1109/TIT.2012.2208937, arXiv:1106.3625] 11

[29] TOM GUR AND ODED LACHISH: On the power of relaxed local decoding algorithms. In *Proc. 31st Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'20)*, pp. 1377–1394. ACM Press, 2020. [doi:10.1137/1.9781611975994.83] 11

[30] TOM GUR, GOVIND RAMNARAYAN, AND RON D. ROTHBLUM: Relaxed locally correctable codes. In *Proc. 9th Innovations in Theoret. Comp. Sci. conf. (ITCS'18)*, pp. 27:1–27:11. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. [doi:10.4230/LIPIcs.ITCS.2018.27] 11

[31] TOM GUR AND RON D. ROTHBLUM: Non-interactive proofs of proximity. *Comput. Complexity*, 27(1):99–207, 2018. Preliminary version in ITCS'15. [doi:10.1007/s00037-016-0136-9] 12, 20

[32] VENKATESAN GURUSWAMI, ATRI RUDRA, AND MADHU SUDAN: *Essential Coding Theory*. 2019. Book. Draft available at author's website. 5, 57

[33] VENKATESAN GURUSWAMI, CHAOPING XING, AND CHEN YUAN: How long can optimal locally repairable codes be? *IEEE Trans. Inform. Theory*, 65(6):3662–3670, 2019. Preliminary version in RANDOM'18. [doi:10.1109/TIT.2019.2891765, arXiv:1807.01064] 11

[34] RICHARD W. HAMMING: Error detecting and error correcting codes. *Bell Sys. Tech. J.*, 29(2):147–160, 1950. [doi:10.1002/j.1538-7305.1950.tb00463.x] 3

[35] BRETT HEMENWAY, NOGA RON-ZEWI, AND MARY WOOTTERS: 2017. Personal communication. 5

[36] JØRN JUSTESEN: Class of constructive asymptotically good algebraic codes. *IEEE Trans. Inform. Theory*, 18(5):652–656, 1972. [doi:10.1109/TIT.1972.1054893] 13

[37] SWASTIK KOPPARTY, OR MEIR, NOGA RON-ZEWI, AND SHUBHANGI SARAF: High-rate locally-correctable and locally-testable codes with sub-polynomial query complexity. *J. ACM*, 64(2):11:1–11:42, 2017. Preliminary version in STOC'16. [doi:10.1145/3051093, arXiv:1504.05653] 5, 9, 10, 43, 44, 45, 47, 48, 50, 57, 58, 61, 62

[38] SWASTIK KOPPARTY AND SHUBHANGI SARAF: Local testing and decoding of high-rate error-correcting codes. *Electron. Colloq. on Comput. Complexity (ECCC)*, TR17-126:1–21, 2017. A version of this paper appeared in SIGACT News. [ECCC] 3

[39] CARSTEN LUND, LANCE FORTNOW, HOWARD J. KARLOFF, AND NOAM NISAN: Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. Preliminary version in FOCS'90. [doi:10.1145/146585.146605] 34

[40] OR MEIR: Combinatorial construction of locally testable codes. *SIAM J. Comput.*, 39(2):491–544, 2009. Preliminary version in STOC'08. [doi:10.1137/080729967] 10, 45

[41] OR MEIR: Combinatorial PCPs with short proofs. *Comput. Complexity*, 25(1):1–102, 2016. Preliminary version in CCC'12. [doi:10.1007/s00037-015-0111-x, ECCC:TR13-134] 6

[42] DANA MOSHKOVITZ AND RAN RAZ: Two-query PCP with subconstant error. *J. ACM*, 57(5):29:1–29:29, 2010. Preliminary version in FOCS'08. [doi:10.1145/1754399.1754402, ECCC:TR08-071] 11

[43] OMER REINGOLD: Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008. Preliminary version in STOC'05. [doi:10.1145/1391289.1391291, ECCC:TR04-094] 45

[44] OMER REINGOLD, GUY N. ROTHBLUM, AND RON D. ROTHBLUM: Constant-round interactive proofs for delegating computation. In *Proc. 48th STOC*, pp. 49–62. ACM Press, 2016. [doi:10.1145/2897518.2897652, ECCC:TR16-061] 35, 38, 39, 45

[45] OMER REINGOLD, SALIL P. VADHAN, AND AVI WIGDERSON: Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Ann. Math.*, 155(1):157–187, 2002. Preliminary version in FOCS'00. [doi:10.2307/3062153, arXiv:math/0406038, ECCC:TR01-018] 45

[46] RONITT RUBINFELD AND MADHU SUDAN: Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. [doi:10.1137/S0097539793255151] 19

[47] CLAUDE ELWOOD SHANNON: Communication in the presence of noise. *Proc. IRE*, 37(1):10–21, 1949. [doi:10.1109/JRPROC.1949.232969] 3

[48] MADHU SUDAN: *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. Springer, 1995. [doi:10.1007/3-540-60615-7] 19

[49] ITZHAK TAMO AND ALEXANDER BARG: A family of optimal locally recoverable codes. *IEEE Trans. Inform. Theory*, 60(8):4661–4676, 2014. Preliminary version in ISIT'14. [doi:10.1109/TIT.2014.2321280, arXiv:1311.3284] 11

[50] PAUL VALIANT: The tensor product of two codes is not necessarily robustly testable. In *Proc. 9th Internat. Workshop on Randomization and Computation (RANDOM'05)*, pp. 472–481. Springer, 2005. [doi:10.1007/11538462_40] 61

[51] ROM R. VARSHAMOV: Estimate of the number of signals in error correcting codes. In *Dokl. Akad. Nauk SSSR (Russian)*, volume 117, pp. 739–741, 1957. 57

[52] MICHAEL VIDERMAN: A combination of testability and decodability by tensor products. *Random Struct. Algor.*, 46(3):572–598, 2013. Preliminary version in RANDOM'12. [doi:10.1002/rsa.20498, arXiv:1105.5806] 61

[53] SERGEY YEKHANIN: Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008. Preliminary version in STOC'07. [doi:10.1145/1326554.1326555] 3

[54] SERGEY YEKHANIN: Locally decodable codes. *Found. Trends Theor. Comp. Sci.*, 6(3):139–255, 2012. [doi:10.1561/0400000030] 3

[55] VICTOR VASILIEVICH ZYABLOV: An estimate of the complexity of constructing binary linear cascade codes (Russian). *Problemy Peredachi Informatsii*, 7(1):5–13, 1971. Link at Mathnet.ru English translation: Probl. Inform. Transmission 7(1) (1971) 3–10. 5

AUTHORS

Tom Gur
Associate professor
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, United Kingdom
tom.gur@warwick.ac.uk
https://www.dcs.warwick.ac.uk/~tomgur/

Govind Ramnarayan
Software Engineer
Neural Magic
Somerville, Massachusetts, USA
govind@neuralmagic@com
http://www.mit.edu/~govind/

Ron D. Rothblum
Assistant professor
Department of Computer Science
Technion
Haifa, Israel
rothblum@cs.technion.ac.il
http://www.cs.technion.ac.il/~rothblum/

ABOUT THE AUTHORS

TOM GUR is an associate professor in the Department of Computer Science at the University of Warwick, and a UKRI Future Leaders Fellow. He received his Ph. D. from the Weizmann Institute of Science, under the supervision of Oded Goldreich, and spent two years at UC Berkeley before joining the University of Warwick. He was awarded the Shimon Even Prize in Theoretical Computer Science. His research interests are primarily in the foundations of computer science and combinatorics. Specific interests include sublinear-time algorithms, complexity theory, coding theory, cryptography, quantum computing, and more.

GOVIND RAMNARAYAN received his Ph. D. from the Massachusetts Institute of Technology in 2020, supervised by Elchanan Mossel. His research interests lie primarily in the intersection of combinatorics and statistics, concretely in applications of these fields to coding theory and algorithmic inference. He currently works as an algorithm designer and software engineer at Neural Magic, a startup in Somerville, Massachusetts, USA, which produces software to accelerate neural network inference on CPUs. In his spare time, he enjoys walking and biking around the Boston area, and trying new restaurants.

RON D. ROTHBLUM is an assistant professor at the department of computer science at the Technion–Israel Institute of Technology. He completed his Ph. D. at the Weizmann Institute of Science in 2015, advised by Oded Goldreich. His dissertation received the John F. Kennedy Ph. D. Distinction Prize and the Shimon Even Prize in Theoretical Computer Science. His research is mainly focused on cryptography and complexity theory.