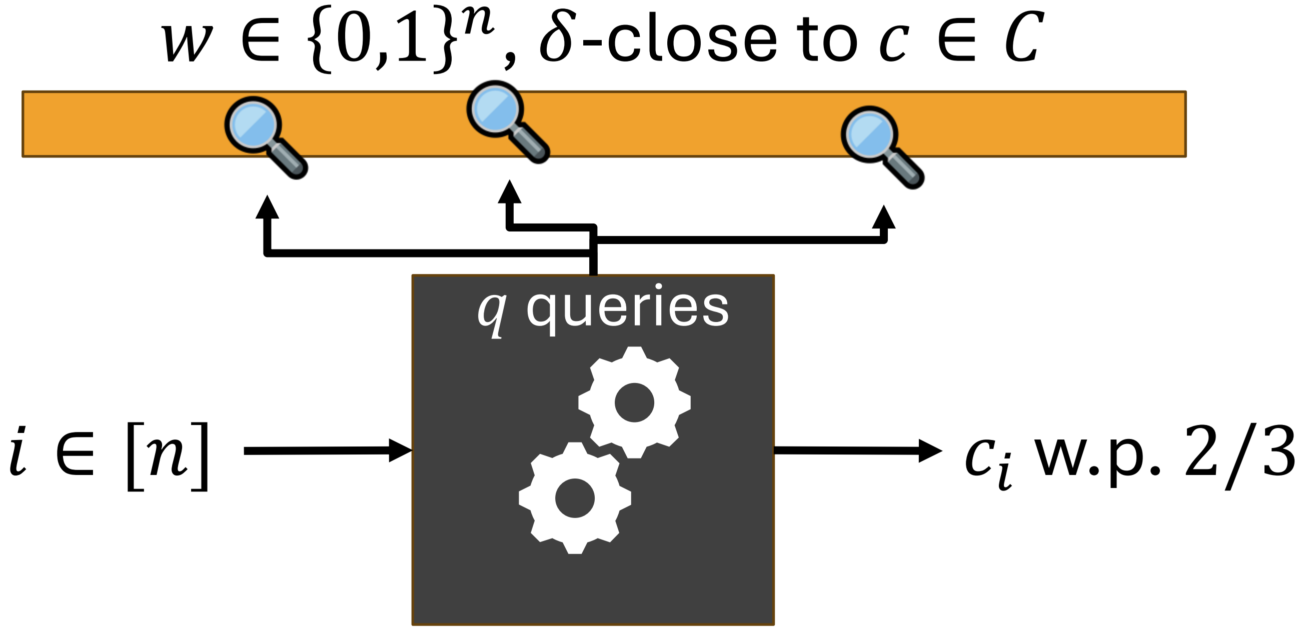
Relaxed Local Correctability from Local Testing

Vinayak M. Kumar and Geoffrey Mon
University of Texas at Austin

Locally correctable codes (LCCs)

LDCs/LCCs have correctors that query very few indices...

...but optimal LCC parameters are a big mystery.



Local correctability: what's known?

Focusing on **asymptotically good** LCCs (constant rate and correcting radius); how many queries required?

Lower bound*

$$q \geq \widetilde{\Omega}(\log n)$$

[Katz–Trevisan 00, Woodruff 07]

Upper bound

$$q \leq 2^{\tilde{O}\left(\sqrt{\log n}\right)} = n^{o(1)}$$

[Kopparty–Meir–Ron-Zewi–Saraf 17]

Best asymptotically good LCCs

Lower bound [Katz–Trevisan 00, Woodruff 07]. $q \geq \widetilde{\Omega}(\log n)$

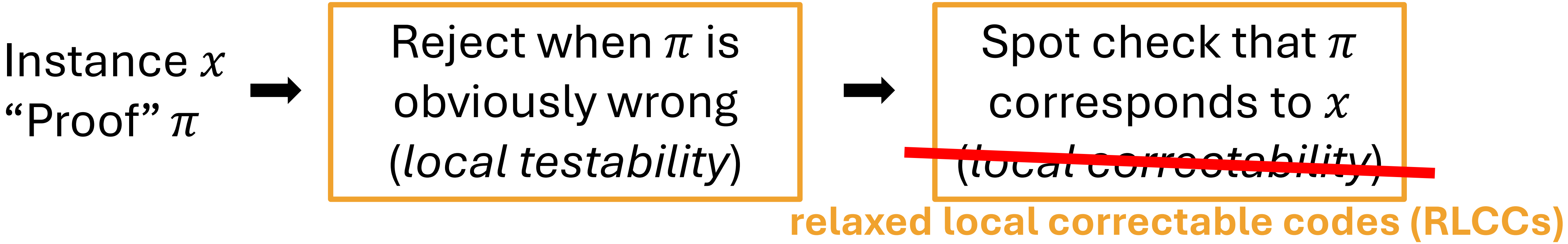
Technique	Queries	Due to
low-degree polynomials	n^{ϵ}	[Babai–Fortnow–Levin–Szegedy 91, Rubinfeld–Sudan 96]
multiplicity codes	n^{ϵ}	[Kopparty–Saraf–Yekhanin 14]
lifted Reed-Solomon	n^{ϵ}	[Guo–Kopparty–Sudan 13]
expander codes	n^{ϵ}	[Hemenway–Ostrovsky–Wootters 15]
distance amplification	$n^{o(1)}$	[Kopparty–Meir–Ron-Zewi–Saraf 17]

high-rate

What if we relax the definition?

Allow the corrector to give up if it detects errors.

Why? e.g. when constructing PCPs, it's convenient if the proofs form a code with good local properties.



Instance x
"Proof" π

➡

Reject when π is obviously wrong (*local testability*)

➡

Spot check that π corresponds to x ~~(*local correctability*)~~

relaxed local correctable codes (RLCCs)

If the local corrector detects errors, we can reject π.

RLCCs: formal definition

Def ([Ben-Sasson–Goldreich–Harsha–Sudan–Vadhan 06, Gur–Ramnarayan–Rothblum 20])**.**
$C \subseteq \{0,1\}^n$ is a *relaxed locally correctable code (RLCC)* with q queries and radius δ if it has a corrector M making q queries s.t.

1. soundness: for all $w \in \{0,1\}^n$ which are δ-close to some $c \in C$,
$$\forall i \in [n].\, \Pr[M^w(i) \in \{c_i, \perp\}] \geq 2/3$$
2. completeness: M never rejects when $w \in C$.

[Ben-Sasson–Goldreich–Harsha–Sudan–Vadhan 06]: when $q = O(1)$, dramatic improvement over known LDC constructions!

Some more applications of RLCCs

PCPs/interactive oracle proofs [Ron-Zewi–Rothblum 20]

Proofs of proximity [Ben-Sasson–Goldreich–Harsha–Sudan–Vadhan 06, Goldreich–Gur–Komargodski 15, Gur–Rothblum 17, Gur–Rothblum 18, Goldreich–Gur 21]

Adaptivity hierarchy for property testing [Cannone–Gur 17]

Fault-tolerant data structures [Chen–Grigorescu–de Wolf 09]

Relaxing helps asymptotically good RLCCs:

Compare to $q \geq \widetilde{\Omega}(\log n)$ for LCCs

Lower bound [Gur–Lachish 21, Dall'Agnol–Gur–Lachish 21, Goldreich 23]. $q \geq \widetilde{\Omega}(\sqrt{\log n})$

Technique	Query complexity	Due to
existing LCCs	$n^{o(1)}$	[Kopparty–Meir–Ron-Zewi–Saraf 17]
iterated tensoring	$(\log n)^{O(\log \log n)}$	[Gur–Ramnarayan–Rothblum 20]
row-evasive partitioning	$(\log n)^{O(\log \log \log n)}$	[Cohen–Yankovitz 22]
nested LTCs	$\log^{69} n$	this work
nested expander codes	$\log^{2+o(1)} n$	[Cohen–Yankovitz 23]

Construction approach

Want RLCC constructions for arbitrarily large block length.

Start with a trivial RLCC with tiny block length.

Boost block length iteratively: use the smaller RLCC to build a bigger one; repeat until desired block length reached.

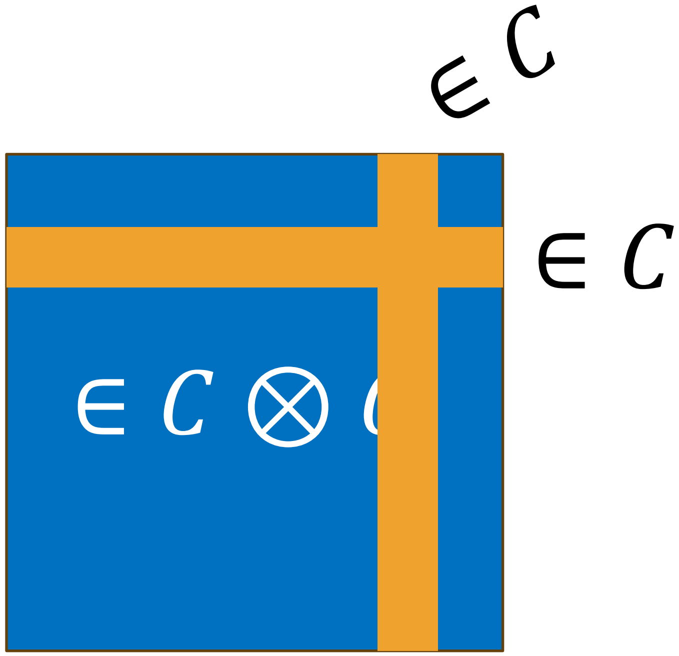
Gur, Ramnarayan, and Rothblum; Cohen and Yankovitz ‘22 follow this approach, and we will too

Prior work: iterated tensor product

[Gur–Ramnarayan–Rothblum] use tensoring to boost block length.

If $C \in \mathbb{F}^n$ is a linear code, then $C \otimes C \in \mathbb{F}^{n \times n}$ is the code where every row and column is in C.

Thm ([Gur–Ramnarayan–Rothblum 20]). If C is an RLCC, then $C \otimes C$ is too.



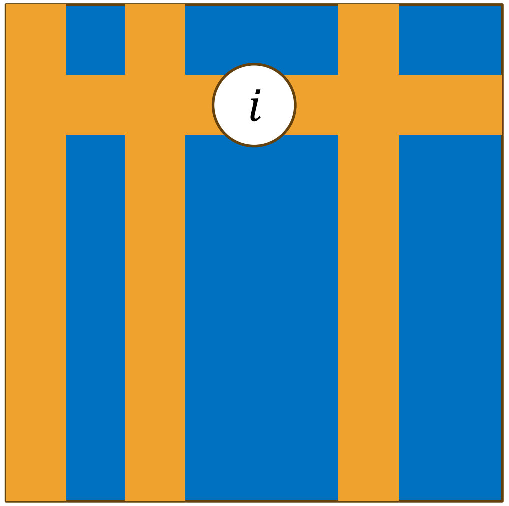
Prior work: tensor product of RLCCs

Run the C-corrector on the row that contains the index that we want.

Use the C-corrector on $\mathrm{polylog}\, n$ random columns
to check that the row is consistent with the rest.

Multiplicative query overhead at each step:
need to recurse on the C-corrector $\mathrm{polylog}\, n$ times.
$\Rightarrow (\mathrm{polylog}\, n) \cdots (\mathrm{polylog}\, n)$
$= (\log n)^{O(\log \log n)}$ total queries

[Cohen–Yankovitz 22] improve query factor per step
to $\mathrm{polylog}(\log n) \Rightarrow (\log n)^{O(\log \log \log n)}$ queries

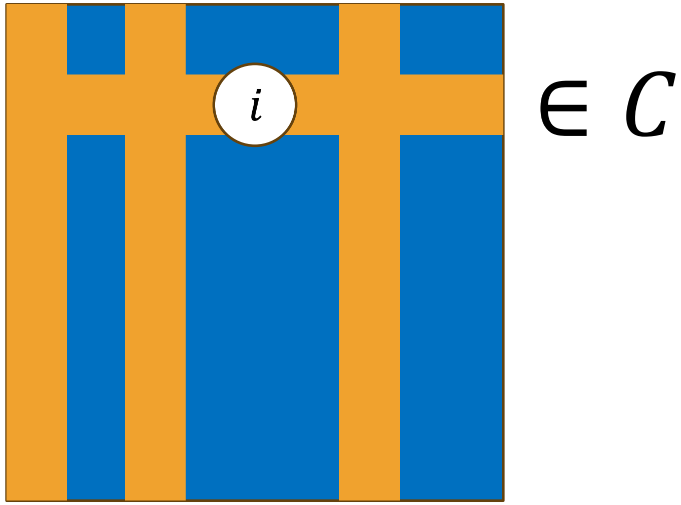


Query cost of tensoring grows too fast

Tensoring compares overlapping parts of the input against each other

At each tensor product step, we must recurse multiple times on the smaller code’s local corrector
⇒ multiplicative query cost 😔

Can we get **additive** query cost? 😳
Instead of recursing multiple times per iteration, let’s use some outside help



🔨: locally testable codes

Locally testable codes (LTCs) can detect corruption locally.

Local tester T makes q queries and rejects with probability proportional to the distance from the code:

$$\forall w \in \{0,1\}^n . \Pr[T^w = \bot] \geq \kappa \cdot \mathrm{dist}(w, \mathrm{LTC}).$$

T never rejects when $w \in \mathrm{LTC}$.

We have great high-rate LTCs [Dinur–Evra–Livne–Lubotzky–Mozes 22]; use them to build iteratively bigger RLCCs!

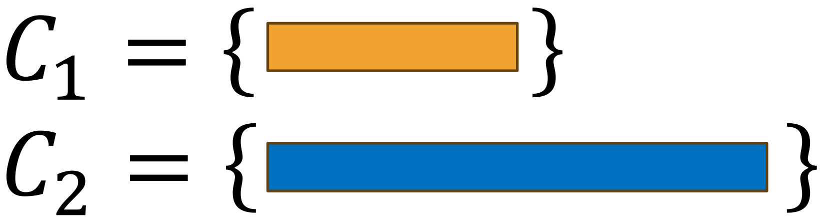
Block length boosting operation: nesting

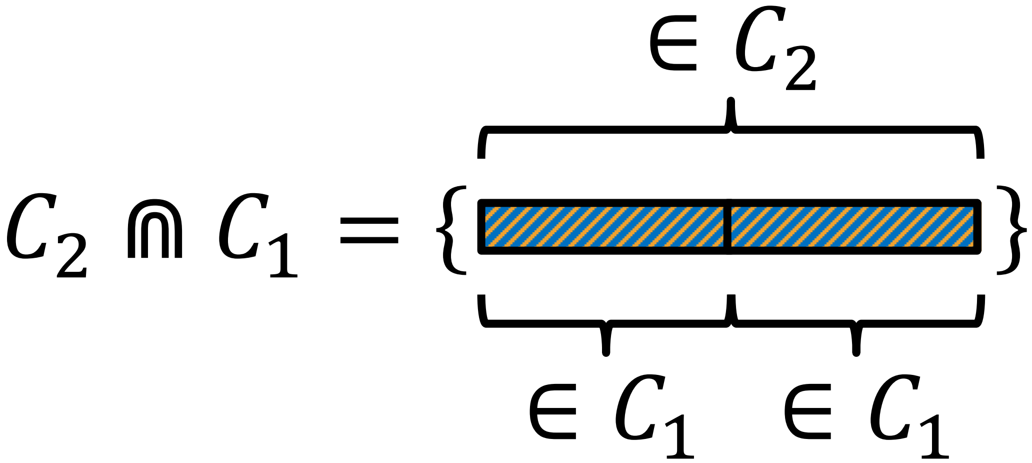
(*n* need not divide *N*)

Def. Let $C_1 \subseteq \{0,1\}^n$ and $C_2 \subseteq \{0,1\}^N$ where n divides N.
The code formed by *nesting* C_1 *in* C_2 is

$$C_2 \Cap C_1 := C_2 \cap C_1^{N/n}.$$

direct product

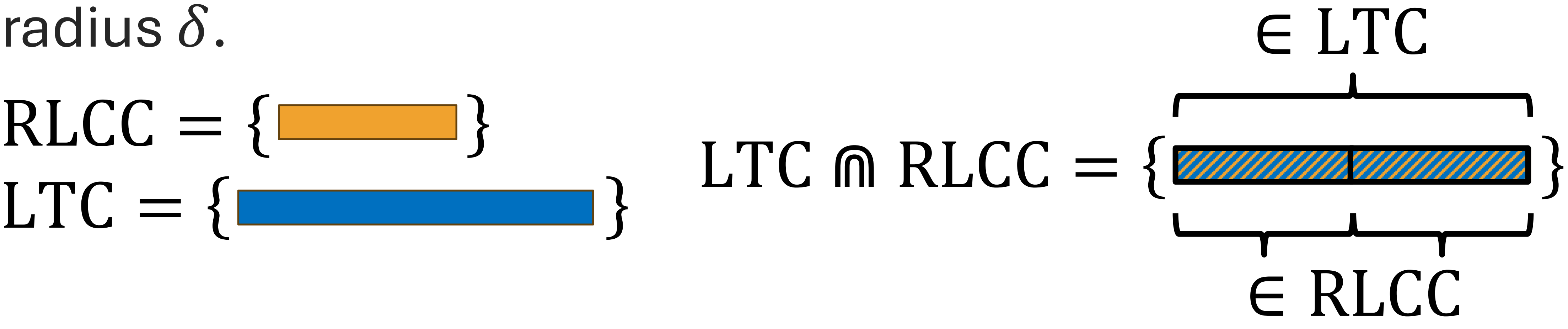




$\mathrm{LTC} \Cap \mathrm{RLCC} \Rightarrow$ bigger RLCC

length N length n length N

Thm. If $\mathrm{RLCC} \subseteq \{0,1\}^n$ has radius δ and $\mathrm{LTC} \subseteq \{0,1\}^N$ has distance 2δ, then $\mathrm{LTC} \Cap \mathrm{RLCC} \subseteq \{0,1\}^N$ is an RLCC with radius δ.



$\mathrm{RLCC} = \{$ ▬ $\}$
$\mathrm{LTC} = \{$ ▬ $\}$
$\mathrm{LTC} \Cap \mathrm{RLCC} = \{$ ▨ $\}$
$\in \mathrm{LTC}$
$\in \mathrm{RLCC}$

Design a local corrector that handles two cases:

- # corrupted bits $\leq \delta n$: *correct* by recursing on smaller RLCC
- # corrupted bits $> \delta n$: *detect* corruption using the tester

Nesting: # corrupted bits $\leq \delta n$

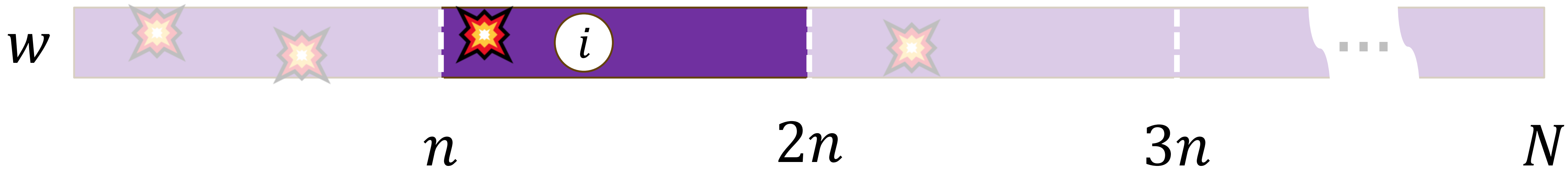
Let input w satisfy $\text{dist}(w, \text{LTC} \Cap \text{RLCC}) \leq \delta$.
Let $c \in \text{LTC} \Cap \text{RLCC}$ be the codeword closest to w.

Close case: suppose # corrupted bits $\leq \delta n$.

Pick the unique interval $I := \{kn + 1, \ldots, kn + n\} \ni i$.

$\text{dist}(w|_I, c|_I) \leq \delta$ and $c|_I \in \text{RLCC}$ by construction.

RLCC has radius δ so we can recursively call its corrector on $w|_I$!



Nesting: # corrupted bits > δn

Let input w satisfy $\mathrm{dist}(w, \mathrm{LTC} \Cap \mathrm{RLCC}) \leq \delta$.
Let $c \in \mathrm{LTC} \Cap \mathrm{RLCC}$ be the codeword closest to w.

Close case: ✅ We can recurse on an interval

Far case: suppose # corrupted bits > δn.

$\Rightarrow \mathrm{dist}(w, c) > \delta n / N$.

$\Rightarrow \mathrm{dist}(w, \mathrm{LTC}) > \delta n / N \Rightarrow$ tester rejects w.p. $\kappa \delta n / N$.

Repeat tester $O(N / \kappa \delta n)$ times to find corruption w.p. $2/3$.

Only an additive query cost!

Let input w satisfy $\mathrm{dist}(w, \mathrm{LTC} \Cap \mathrm{RLCC}) \le \delta$.
Let $c \in \mathrm{LTC} \Cap \mathrm{RLCC}$ be the codeword closest to w.

Close case: ✅ We can recurse on an interval

Far case: ✅ Local tester finds corruption

Putting it together: the local corrector for $\mathrm{LTC} \Cap \mathrm{RLCC}$

Run the LTC tester $O(N/\kappa\delta n)$ times

Recurse on the RLCC corrector for the interval containing i

⇒ Nesting adds only $O(N/\kappa\delta n) \cdot q_{\mathrm{LTC}}$ more queries

Nesting: parameters summary

LTC	RLCC	$\Rightarrow$	$\mathrm{LTC} \Cap \mathrm{RLCC}$
length N	length n		length N
distance 2δ	correcting radius δ		correcting radius δ
q_{LTC} queries	q queries		$q + O(q_{\mathrm{LTC}} N / \kappa \delta n)$ queries
rate $1 - \varepsilon_{\mathrm{LTC}}$	rate $1 - \varepsilon$		rate $1 - \varepsilon - \varepsilon_{\mathrm{LTC}}$
LTC has $\varepsilon_{\mathrm{LTC}} N$ linear constraints	$\mathrm{RLCC}^{N/n}$ has εN linear constraints		at most $(\varepsilon + \varepsilon_{\mathrm{LTC}})N$ linear constraints

Iteratively nesting to build RLCCs

Well-behaved: nesting incurs only **additive** cost in rate, queries.

Now we can iterate:

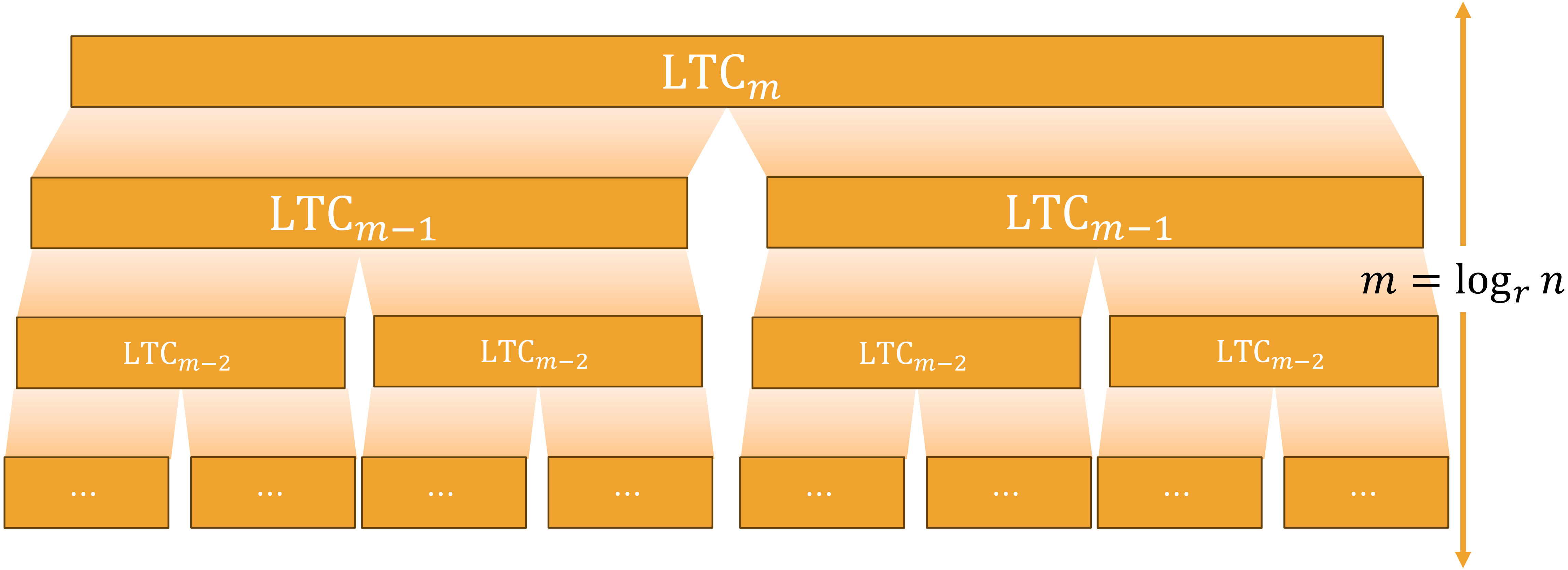
$$C \coloneqq \mathrm{LTC}_m \Cap \left(\mathrm{LTC}_{m-1} \Cap (\dots(\mathrm{LTC}_2 \Cap \mathrm{LTC}_1)\dots)\right)$$

Let each $\mathrm{LTC}_j \subseteq \{0,1\}^{n_j}$ have rate $1-\varepsilon$, distance 2δ, q queries, and suppose $n_{j+1}/n_j \ = r$ for all j.

Then C is an RLCC with rate $1 - m\varepsilon$, radius δ, and query complexity

$$n_1 + \sum_j O\left(\frac{qn_{j+1}}{\kappa\delta n_j}\right) = n_1 + O\left(\frac{mrq}{\kappa\delta}\right).$$

Iteratively nesting: visual



Iteratively nesting: concrete parameters

C is an RLCC with rate $1 - m\varepsilon$, radius δ, and query complexity

$$n_1 + O\left(\frac{mrq}{\kappa\delta}\right).$$

C needs constant rate ⇒ pick an LTC family with $\varepsilon = O(1/\log n)$.

Thm ([Dinur–Evra–Livne–Lubotzky–Mozes 22]). 🔨

For any $\varepsilon \in (0,1)$, there is a family of explicit linear LTCs with $R = 1 - \varepsilon$; $q, r = \mathrm{poly}(1/\varepsilon)$; and $\delta, \kappa = \mathrm{poly}(\varepsilon)$.

$\Rightarrow \frac{mrq}{\kappa\delta} = \mathrm{polylog}\, n$ 🎉

Iteratively nesting: almost done!

After plugging in [Dinur–Evra–Livne–Lubotzky–Mozes 22],

$C \coloneqq \mathrm{LTC}_m \Cap \left(\mathrm{LTC}_{m-1} \Cap (\dots(\mathrm{LTC}_2 \Cap \mathrm{LTC}_1)\dots)\right)$ is an RLCC with

Rate $1 - \frac{\log_r n}{\log n} = 1 - o(1)$ ✅ because $r = \operatorname{polylog} n$,

query complexity $\operatorname{polylog} n$ ✅, and

distance $1/\operatorname{polylog} n$ 😔

Thm. Asymptotically good RLCCs with query complexity $\log^{69} n$.

Closing remarks

Cohen and Yankovitz improve queries to $\log^{2+o(1)} n$

Optimal query complexity: $\log n$ or even $\sqrt{\log n}$?

RLCC with (poly)log queries which is also an LTC?

{vmkumar,gmon}@cs.utexas.edu

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-2137420. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.