The QEC Challenge

A public, automatically verified leaderboard for quantum low-density parity-check codes

Unitary Foundation `github.com/unitaryfoundation/qldpc-challenge`

July 30, 2026

Abstract

The QEC Challenge is a public leaderboard for quantum low-density parity-check codes. A participant submits a single JSON file describing one CSS code; continuous integration recomputes the code's properties and actively tries to refute its distance claim, and only a code that holds up is merged onto the board. Three principles shape the design. (i) *Everything cheap is trustless and mandatory*: the verifier recomputes n, k, CSS commutation, check weight, and the geometric locality class from scratch over $\mathbb{F}_2$ — a wrong claimed k or a non-commuting "stabilizer" is rejected in milliseconds. (ii) *Distance is layered by how much is proven*: a submitter attaches an explicit logical-operator *witness* that proves $d \leq$ value with no trust required; an adversarial *refutation gate* then searches for a lighter logical, and separate certification tiers upgrade a claim to *corroborated* or *exact*. (iii) *A code is not a single number*: the primary tracks are a grid of two *computed* axes (locality class × check-weight class), and within each cell the ranking is a Pareto frontier over (n, k, d, w) rather than a scalar; two efficiency figures — the *operational* efficiency kd^2/n and, for codes shipping a verified layout, the *geometric* efficiency $g = 4kd^2/(n\rho^2 r^4)$, normalized so the planar surface code scores exactly 1 — are reported alongside the frontier, never in place of it. The whole pipeline — validator, refutation gate, exact certifier, scorer, static-site generator — lives in the repository and runs in CI on every pull request, which also makes the board usable as the verification harness of automated (LLM- and agent-driven) code-discovery loops.

1 Overview

A submission is one JSON file placed under `codes/`, conforming to `schema/code.schema.json` (described in `schema/SCHEMA.md`). A contributor opens a pull request adding the file (one new code per PR, enforced by CI); GitHub Actions runs the verifier (`verify/qldpc_verify.py`) and the distance refutation gate (`verify/gate_changed.py`), and a green check means the code's cheap properties are confirmed *and* an adversarial search failed to find a logical lighter than the claimed distance. After merge, `site/build.py` regenerates the static leaderboard into `docs/` automatically (an index with per-cell Pareto tables and plots, a page per code, a reference list, and status badges). The board is seeded with literature baselines — the planar open-boundary codes of Liang, Eberhardt, and Chen [1], the bivariate-bicycle gross code [2], generalized-bicycle codes [3, 4], coset codes [5], and the surface, toric, and Steane codes [6, 7] — so a new entry is always measured against published codes rather than an empty board.

Two things are deliberately *not* asked of the submitter: a proof that their distance is exact (that is the server's job), and a single headline rank (the board is a Pareto frontier). Everything else — the parity checks, the claimed n, k, d with witnesses, an optional layout, and provenance — is supplied and then recomputed, witnessed, or attacked by the verifier.

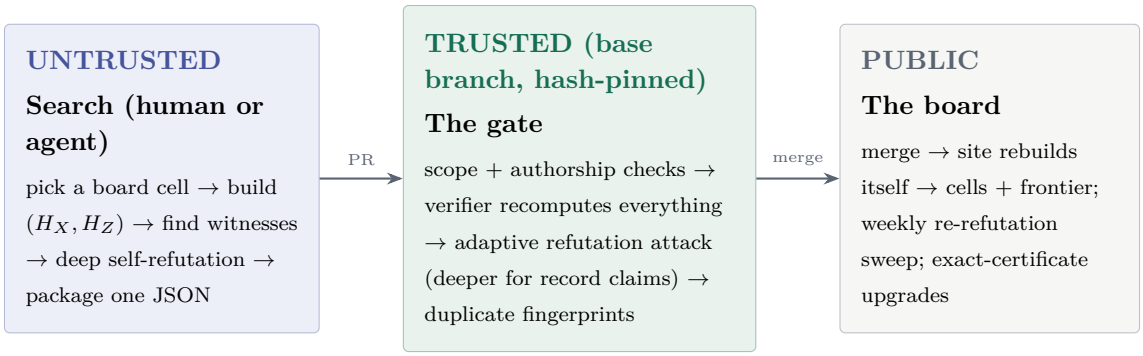

Figure 1: Three zones, one direction of trust. A submission originates in an untrusted search (increasingly, an automated one), is judged entirely by pinned code running from the base branch, and only then becomes part of the public record — which keeps being re-attacked after merge.

2 Why a leaderboard of cells, not one number

A quantum code trades several quantities against one another — physical qubits n, logical qubits k, distance d, maximum check weight w, and geometric locality — and separately has a decoding performance. Collapsing all of that into one scalar would reward gaming one axis at the expense of the rest. So the board is stratified in three layers (`TRACKS.md`).

Layer 1: primary tracks, computed, never self-declared. The primary tracks are a grid of two nested axes whose membership is *derived by the verifier* from the parity checks and the layout, so a track cannot be claimed by relabeling:

- a **locality class**: `local-2d-single` (one layer, interaction radius ≤ 4) $\subset$ `local-2d-bilayer` (up to two layers — the flip-chip regime — radius ≤ 7) $\subset$ `unrestricted` (everything else, including any code without a layout);
- a **check-weight class**: `weight-4` $\subset$ `weight-6` $\subset$ `weight-8` $\subset$ any weight, from the maximum row weight of H_X, H_Z.

The classes nest: a single-layer weight-4 code also competes on every looser board. Each (locality, weight) cell is a board of its own (Figure 2).

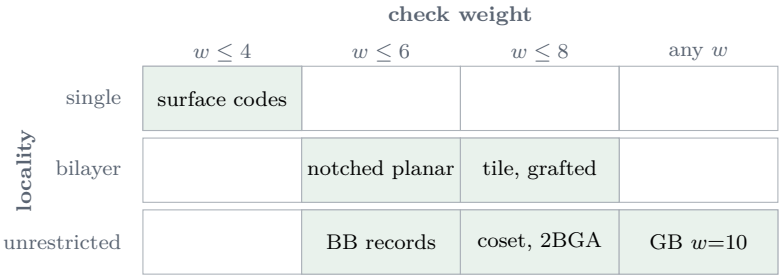

Figure 2: The primary-track grid: locality class × check-weight class, both computed from the submission (example occupants shown). Within each cell the ranking is the Pareto frontier over (n, k, d, w); a code in a tight cell also competes in every looser cell it nests into.

Layer 2: family tags, filterable, never ranked. The construction family (bivariate bicycle, generalized bicycle — including non-abelian two-block group-algebra constructions [8] — coset, tile,

topological, ...) cannot be recovered from the matrices, so it is a self-reported, filterable tag that confers no ranking advantage.

Layer 3: verified flags. Only properties a verifier or a certificate can prove: CSS commutation, the locality class (which doubles as Layer-1 membership), and the distance-confidence tier of Section 6.

Pareto frontier, not a single winner. Within a cell, a code is a record if no other code in the cell dominates it — no other has $n' \le n$, $k' \ge k$, $d' \ge d$, $w' \le w$ with at least one strict inequality. There can be many co-leaders.

Efficiency as sortable columns, not the rank. Two efficiency figures are reported per cell and sortable, but neither collapses the frontier into one number: the *operational* efficiency kd^2/n used by the 2D-locality literature, and the *geometric* efficiency g of Section 3.1, which additionally prices the layout a code ships with. A code whose distance is only an upper bound inherits a $\le$ marker on both figures.

3 Background: two efficiency metrics and the BPT bound

For an $[[n, k, d]]$ stabilizer code, the Bravyi–Poulin–Terhal (BPT) bound [9] states that any geometrically 2D-local code obeys

$$\frac{kd^2}{n} \;\le\; c(r, w, q), \tag{1}$$

where the constant c depends only on the interaction range r, the maximum stabilizer weight w, and the on-site dimension q — not on n. The optimal constant is open, which makes "within a fixed locality model, maximize kd^2/n" a well-posed target for the 2D-local cells. For asymptotically good codes [10, 11], where k and d both grow linearly with n, kd^2/n grows like n^2 without bound — which is exactly why the board buckets and compares like with like rather than ranking a $[[120, \dots]]$ entry against a $[[10^4, \dots]]$ one. Reference points from the literature, spanning the efficiency range (all $q = 2$; w = max check weight):

Code	$[[n, k, d]]$	topology	w	kd^2/n
Rotated planar surface [6]	$[[d^2, 1, d]]$	planar	4	1
Gross code (IBM) [2]	$[[144, 12, 12]]$	periodic	6	12
Tile code [12]	$[[512, 18, 19]]$	planar	8	12.7
Generalized bicycle [3]	$[[126, 28, 8]]$	periodic	10	14.2
Coset two-block [5]	$[[168, 20, 14]]$	unrestricted	8	23.3

The planar rows are the open-boundary program of Liang, Eberhardt, and Chen [1] and the tile codes [12]; the coset row is the two-block generalization of Aydin, Tamo, and Barg [5]. Whether planar codes can close the gap to the periodic and unrestricted records is the kind of question a live leaderboard can resolve.

3.1 Geometric efficiency: pricing the layout

Operational efficiency deliberately ignores *how* a code is laid out: a periodic code and a planar code with the same $[[n, k, d]]$ score the same kd^2/n, although one assumes long-range couplers the

other does not need. The board therefore also reports, for every code whose 2D layout the verifier has accepted, a *geometric* efficiency that prices the layout itself:

$$g \;=\; \frac{4\,kd^2}{n\,\rho^2\,r^4}, \tag{2}$$

where r is the *measured* interaction radius of the shipped layout (the maximum Euclidean check diameter, in units of the unit qubit spacing) and ρ its layer count. This is the BPT ratio (1) charged for its geometric resources: the r^4 factor is the range dependence of the BPT constant in 2D, and ρ^2 is the unique exponent that makes re-packaging a layout into layers score-neutral, so g cannot be inflated by stacking. The constant $4 = (\sqrt{2})^4$ normalizes the rotated planar surface code ($r = \sqrt{2}$, $\rho = 1$) to exactly $g = 1$, which gives the figure an absolute meaning: a code family that sustains $g > 1$ at growing distance would beat the surface code at its own game — more protected logical content per qubit under the *same* hardware locality budget. No known code does this, so unlike the open-ended kd^2/n column, g has a meaningful ceiling that the board makes visible.

Three design details keep g honest. It is computed only from a verifier-accepted layout (Section 5, V8) — r and ρ are recomputed from the shipped coordinates, never taken from the submission — and a code without a layout simply has no g, which is a certification status rather than a judgment. Constant-distance tilings exceed $g = 1$ trivially (a $[[4, 2, 2]]$ block on one plaquette scores $g = 2$), so the board's headline figure requires $d \geq 3$. And a code whose distance is an upper bound inherits a $\leq$ on g as well. At the time of writing the rotated surface codes hold the ceiling at $g = 1$ exactly; the best contributed layouts reach $g \approx 0.38$, and whether *any* honestly laid-out qLDPC code can approach or beat the surface-code ceiling is one of the sharpest open questions the board poses.

4 Submission format

A submission is one JSON object (`schema_version "0.1"`, `code_type "CSS"`) with:

- `n` — physical qubit count; must match the indices used in `checks`.
- `k` — *claimed* logical qubit count; the verifier recomputes it and requires an exact match.
- `checks.X`, `checks.Z` — the parity checks as sparse supports: each is a list of checks, each check a sorted list of distinct 0-based qubit indices.
- `distance.d` — claimed distance $= \min(d_X, d_Z)$, and per side `distance.X`/`distance.Z`, each a {`value`, `confidence`, `witness`} triple, where `confidence` is `"upper_bound"` or `"exact"` and `witness` is the support of a logical operator of that Pauli type and weight `value`.
- `locality` (optional) — `coordinates` (one $[x, y]$ per qubit), `layers`, and a claimed interaction radius; the verifier recomputes the radius and derives the locality class from it. A missing or dishonest layout is not an error — the code simply competes as `unrestricted`.
- `provenance` — `authors` (the humans; the PR author must be listed), `construction` (a reproducible recipe), optional `references`, `date`, `notes`, an optional literature-`novelty` status (`known_parameters`/`new_parameters`, a submitter claim), and an optional `model` field naming the AI system that produced the code, to a specific version — machine-discovered entries are attributed as such.

(A legacy self-declared `tracks` field is deprecated and ignored for ranking; track membership is computed.) Verify locally before opening the PR (the project uses `uv`):

```
uv run python verify/qldpc_verify.py codes/your-code.json
```

The verifier prints a JSON report — per-check pass/fail, the computed n, k, ranks, weight class, locality class — and an `earned_distance` block giving the tier each side actually earned, and exits

0 iff every required check passes. The `cli/` launcher (`./qldpc submit`) automates the whole path from parity-check matrices (however constructed — e.g. with the `qLDPC` library [13]) to a verified, PR-ready file.

5 What the verifier checks

A submission is rejected unless all of the following hold. Checks (V1)–(V9) are polynomial-time and run in CI on every PR; the refutation gate and the certification tiers (Section 6) are layered on top.

(V1) **Schema and indices.** The document conforms to `code.schema.json`, every qubit index in `checks` is in $[0, n)$, and the file is within size limits.

(V2) **No collapsed checks.** No qubit index is repeated within a single check (a repeat would XOR away and silently change the code).

(V3) **CSS commutation.** $H_X H_Z^\top = 0$ over $\mathbb{F}_2$.

(V4) **Logical dimension.** The verifier recomputes $k = n - \mathrm{rank}_{\mathbb{F}_2}(H_X) - \mathrm{rank}_{\mathbb{F}_2}(H_Z)$, requires it to equal the claimed k, and requires $k \geq 1$.

(V5) **LDPC weight cap and vocabulary.** The maximum check weight is bounded (the board is for LDPC codes) and the family tag is from the controlled vocabulary.

(V6) **Distance witnesses.** For each provided side, the witness v must (a) have weight equal to the claimed `value`, (b) lie in the kernel of the opposite checks (an X-witness commutes with H_Z), and (c) be nontrivial, i.e. not in the rowspace of its own checks (not a stabilizer product). This certifies $d_{\text{side}} \leq$ `value` as an *upper bound* with no trust required.

(V7) **Distance consistency.** The claimed d equals the minimum over the witnessed sides.

(V8) **Locality class (if a layout is present).** `coordinates` must cover all n qubits; at most `layers` qubits may share a site and distinct sites must be ≥ 1 apart (no cramming — a small radius cannot be faked by collapsing qubits onto a point); the recomputed maximum check diameter must be within the class cap. The class is *earned* from the layout, never declared.

(V9) **Quick refutation.** A short random-information-set search must fail to find a logical lighter than the claim (the full gate of Section 6 runs far deeper).

Every `exact` confidence claim is *downgraded to upper_bound here* and flagged for server certification — the cheap PR check never upgrades a distance to exact.

6 Distance: three tiers and an adversarial gate

The distance is the minimum weight over nontrivial logicals; for a CSS code $d = \min(d_X, d_Z)$. The problem is NP-hard [14], and the hardness is asymmetric: exhibiting a low-weight logical gives only an *upper* bound, while certifying $d \geq D$ is the hard direction — and a submitter has every incentive to overclaim. Heuristic distance estimates routinely deflate under deeper search, so the challenge layers three confidence tiers (Figure 3) behind an active gate.

Upper-bound tier (trustless, in CI). The submitter's witness is checked exactly as in (V6). A valid witness proves $d_{\text{side}} \leq$ `value`; the board labels such a code $d \leq$ and lets anyone climb instantly, since the arithmetic is checked rather than trusted.

The refutation gate (adversarial, in CI). Every changed code faces independent bounded searches for a logical *lighter* than the claim: a random-information-set (RIS) search (randomized distance estimation in the spirit of `QDistRnd` [15]) and a syndrome-decoder (BP+OSD) search [16]

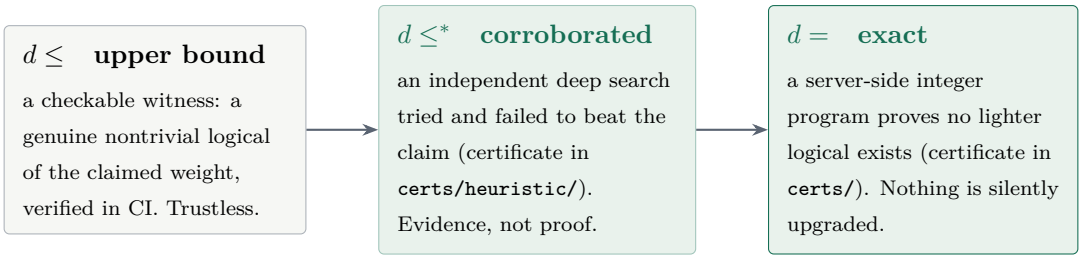

Figure 3: The distance-confidence ladder. Every tier is backed by an artifact in the repository: a witness inside the submission, a heuristic certificate, or an exact certificate.

— two genuinely different mechanisms. Budgets are adaptive: trials scale with n, and a code that would enter a cell's Pareto frontier faces a much deeper multi-seed battery than a dominated one, so over-claims pay extra scrutiny exactly where gaming would matter. Frontier claims additionally face an accelerated RIS pass (a bit-packed $\mathbb{F}_2$ engine at roughly 150× the baseline trial budget) that only *proposes* candidates: a find counts as a refutation solely after the trusted verifier validates the witness, so the extra depth adds no trust surface. Any hit is a sound refutation — it exhibits a checkable counterexample — and fails the PR. The per-run seed is random by design, so an over-claim cannot be tuned to evade one fixed search; a weekly whole-board sweep re-runs the refutation with fresh seeds to accumulate coverage over time.

Corroborated tier. A maintainer-run deep search (`verify/heuristic_certify.py`) that tries and fails to beat a claim writes a heuristic certificate; the board then shows $d \leq^*$. This is evidence, not proof — the label records exactly how hard the claim has been attacked.

Exact tier (server-certified). To upgrade a side to $d =$, a maintainer runs `verify/certify.py`, which proves no lighter nontrivial logical exists via a bounded integer program (scipy/HiGHS, no commercial solver). For each logical generator t_L of the relevant type it asks whether a v exists with

$$Hv \equiv 0, \qquad \langle v, t_L \rangle \equiv 1 \pmod 2, \qquad \mathrm{wt}(v) \leq \texttt{value} - 1, \tag{3}$$

the mod-2 constraints linearized by integer slacks. If every generator on both sides is proven *infeasible*, d is exact; if a lighter logical is found, the claim is *refuted*; if the solver hits its time limit, the side honestly remains an upper bound. A successful run writes a certificate to `certs/<slug>.json`, which is what upgrades a code's board label to $d =$. Exact records outrank equal upper-bound ones; nothing is ever silently upgraded.

7 Trust architecture and automated discovery

Automated code discovery already works: LLM-guided evolutionary search [17], reinforcement learning [18], and LLM-driven concept evolution over non-abelian families [19] all produce candidate codes far faster than humans can referee them — and an overstated distance is precisely their characteristic failure mode. Several of the board's current records were machine-discovered and are attributed to the producing model in their provenance. The pipeline is therefore built so that *nothing a submission can contain is trusted*, which is what makes it usable as the verification harness of an autoresearch loop (Figure 1):

- **Scope separation.** A PR that adds code data may not touch the verifier, schema, workflows, or site builder (`check_submission_scope.py`); and for code PRs the entire judgment runs from the *base-branch* checkout, so a submission cannot alter the code it is judged by.
- **Pinned validation stack.** The trusted closure — the verifier, the refutation engine, the GF(2) core, the local `validate_candidate` gate, and the integrity checker itself — is pinned by a SHA-256 manifest; any drift fails CI, and re-pinning is a deliberate, reviewable act.
- **Authorship binding.** The PR author must be a listed author of the codes they add (`check_authorship.py`).
- **Duplicate fingerprints.** The reduced-row-echelon forms of (H_X, H_Z) pin the exact stabilizer group — an identical fingerprint is a hard CI error — and a permutation-invariant Weisfeiler–Leman signature on the Tanner graph flags possible equivalents for human review.
- **Local gate for agents.** An autoresearch agent may explore freely, but by convention (`research/AUTORESEARCH.md`) no code is a find until the pinned `validate_candidate` returns `passed: true` — the same verdict CI will reach, computable before a PR is ever opened.

8 Repository layout and automation

Path	Contents
`schema/`	submission format (JSON Schema + `SCHEMA.md`)
`verify/`	the trusted stack (`gf2.py`, `qldpc_verify.py`, `heuristic_distance.py`, `validate_candidate.py`, all hash-pinned); the CI gates (`gate_changed.py`, scope, authorship, integrity); the offline tiers (`certify.py`, `heuristic_certify.py`); an optional C++ search accelerator; tests
`decode/`	the BP+OSD syndrome-decoder cross-check
`codes/`	accepted submissions and seeded baselines (the board data)
`certs/`	exact and heuristic distance certificates
`research/`	contributor tooling: family samplers and screening kit, the open-boundary planar engine, and the autoresearch agent manual
`site/`	`build.py`, which generates the static site
`docs/`	the generated site (index, per-code pages, references, badges)
`cli/`	`./qldpc submit`: matrices in, verified PR-ready JSON out
`TRACKS.md`	the three ranking layers and how cells work

CI (`.github/workflows/verify.yml`) runs, in order: the scope check, the validator-integrity check, the convention-discovered self-tests, `verify_all.py` over every board code, the refutation gate on the changed codes, and the authorship check. A separate weekly workflow re-attacks the whole board with fresh seeds (`refute-weekly.yml`), and the site workflow rebuilds `docs/` on every push to `main`. The badges and figures are regenerated from the live board on every build, so they track the current numbers rather than a hand-typed count.

State of the board. At the time of writing the board holds 151 verified codes — 75 submitted through the challenge and 76 literature baselines, 29 of them certified exact, 44 shipping a verifier-accepted 2D layout — across the bivariate-bicycle, generalized-bicycle (including non-abelian two-block group-algebra), coset, tile, and topological families. Machine-discovered entries illustrate the intended dynamics. The best operational efficiency is $kd^2/n = 303.6$, held by a $[[390, 82, 38]]$ generalized-bicycle code over $\mathbb{Z}_{195}$ (weight-32 checks, so it competes only in the any-weight cell;

distance at the upper-bound tier) attributed to the producing LLM in its provenance. The weight-capped cells tell a more constrained story — a $[[684, 8, 85]]$ code leads weight-8 at $kd^2/n = 84.5$ and a $[[360, 12, 24]]$ code leads weight-6 at $kd^2/n = 19.2$ — and the 2D-local cells a more constrained one still: a $[[263, 16, 12]]$ grafted planar bilayer code leads the bilayer weight-8 cell at $kd^2/n = 8.76$. On the geometric side the rotated surface codes hold the $g = 1$ ceiling exactly and the best contributed layouts reach $g \approx 0.38$, so the surface code remains unbeaten once locality is priced in. The board's first odd-k entry, a $[[240, 13, 15]]$ code on the symmetric group S_5, stands (odd k is impossible for abelian two-block constructions). Each entry survived the same discipline the gate enforces — deep multi-seed self-refutation before submission, with witnesses validated by the pinned stack — and in one agent-driven sweep, five of seven staged frontier candidates were refuted *before* submission. The live figures are in `docs/stats.json`.

9 Open questions this could surface

- Can planar (open-boundary) codes close the efficiency gap to the periodic and unrestricted records, or is part of the gap intrinsic to boundaries?
- Can any honestly laid-out qLDPC code beat the surface code's geometric-efficiency ceiling $g = 1$ at growing distance? The best contributed layouts sit at $g \approx 0.38$, and the r^4 price on interaction radius is what every long-range construction must pay down.
- How far does the non-abelian two-block design space reach? Groups outside the systematically enumerated ranges — the board's $\mathrm{PSL}(2, 7)$ and S_5 entries are first examples — keep yielding record parameters.
- How deep must a refutation search run before an upper-bound claim at a given n deserves confidence? The board's history, including refutations of its own staged candidates, is a growing dataset on exactly this question.
- Can the exact-certification tier scale past $n \approx 300$, where the integer programs currently time out?

References

[1] Zijian Liang, Jens Niklas Eberhardt, and Yu-An Chen. Planar quantum low-density parity-check codes with open boundaries. 2025.

[2] Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J. Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627:778–782, 2024.

[3] Pavel Panteleev and Gleb Kalachev. Degenerate quantum LDPC codes with good finite length performance. *Quantum*, 5:585, 2021.

[4] Renren Wang and Leonid P. Pryadko. Distance bounds for generalized bicycle codes. *Symmetry*, 14(7):1348, 2022.

[5] Arda Aydin, Itzhak Tamo, and Alexander Barg. Breaking the bicycle frame: Coset-based quantum LDPC codes, 2026.

[6] A. Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[7] Andrew M. Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793–797, 1996.

[8] Hsiang-Ku Lin and Leonid P. Pryadko. Quantum two-block group algebra codes. *Physical Review A*, 109(2):022407, 2024.

[9] Sergey Bravyi, David Poulin, and Barbara Terhal. Tradeoffs for reliable quantum information storage in 2D systems. *Physical Review Letters*, 104(5):050503, 2010. arXiv:0909.5200.

[10] Pavel Panteleev and Gleb Kalachev. Asymptotically good quantum and locally testable classical ldpc codes. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 375–388, 2022.

[11] Jean-Pierre Tillich and Gilles Zémor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2014.

[12] Vincent Steffan, Shin Ho Choe, Nikolas P. Breuckmann, Francisco Revson Fernandes Pereira, and Jens Niklas Eberhardt. Tile codes: High-efficiency quantum codes on a lattice with boundary, 2025.

[13] Michael A. Perlin. qLDPC. https://github.com/qLDPCOrg/qLDPC, 2023.

[14] Upendra Kapshikar and Srijita Kundu. On the hardness of the minimum distance problem of quantum codes. *IEEE Transactions on Information Theory*, 69(10), 2023. arXiv:2203.04262.

[15] Leonid P. Pryadko, Vadim A. Shabashov, and Valerii K. Kozin. QDistRnd: A GAP package for computing the distance of quantum error-correcting codes. *Journal of Open Source Software*, 7(71):4120, 2022.

[16] Joschka Roffe, David R. White, Simon Burton, and Earl Campbell. Decoding across the quantum low-density parity-check code landscape. *Physical Review Research*, 2(4):043423, 2020.

[17] Juan Cruz-Benito, Andrew W. Cross, David Kremer, and Ismael Faro. Evolutionary discovery of bivariate bicycle codes with llm-guided search, 2026.

[18] Austin Yubo He and Zi-Wen Liu. Discovering highly efficient low-weight quantum error-correcting codes with reinforcement learning, 2025. arXiv:2502.14372.

[19] Zidu Liu and Florian Marquardt. Large-language-model discovery of quantum LDPC codes through structured concept evolution, 2026. arXiv:2606.24808.