AUTOMATED DISCOVERY OF QUANTUM SUBSYSTEM CODES WITH AN AI SCIENTIST

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantum error correction (QEC) is essential for scalable fault-tolerant quantum computing, yet practical code design remains constrained by the connectivity and imperfections of the underlying hardware. This challenge is particularly acute for superconducting processors, where fixed and sparse couplings can restrict both the realizable code space and the level of logical protection that can be achieved. Here we introduce a verification-feedback-driven LLM framework for hardware-constrained subsystem-code discovery. The framework uses an LLM to generate hardware-aware code constructions, deterministic tools to verify them, and verified feedback to improve subsequent generations. Its central feature is that code generation is conditioned on the specific hardware instance, and independently verified code properties and failure evidence are fed back to guide subsequent refinement. We evaluate the framework on four calibration-informed defect layouts of the 105-qubit Willow graph and three connectivity-design tasks: two Willow augmentations and one free-connectivity search within the same qubit budget. On the four defect layouts, the selected subsystem implementations use weight-two gauge measurements and attain larger code and circuit fault distances than native surface-code templates, while reducing CZ count by 12.5%–41.2% and CZ depth by 70.0%–86.1% relative to code-distance-matched routed implementations. Under Willow augmentation, the search finds a distance-7 subsystem code protecting two logical qubits at maximum degree 5 and a distance-5 stabilizer code ($r = 0$) protecting 10 logical qubits at maximum degree 6. A separate free-connectivity search finds a distance-9 subsystem construction at maximum degree 4. These results demonstrate the potential of self-evolving LLM search for QEC design under realistic hardware constraints, paving a path toward robust and resource-efficient fault-tolerant quantum computing.

1 INTRODUCTION

Quantum computing is moving beyond the noisy intermediate-scale quantum (NISQ) era, as advances in quantum error correction (QEC) bring several hardware platforms closer to early fault-tolerant operation (Preskill, 2018; Campbell, 2024; Bluvstein et al., 2024; Google, 2025). Superconducting quantum processors stand out as one of the most mature platforms, with large-scale integrated devices and recent demonstrations of below-threshold logical memories (De Leon et al., 2021; Google, 2023; 2025). However, a major limitation is their fixed and relatively sparse connectivity: after fabrication, two-qubit gates are largely restricted to predefined couplers between nearby qubits (Versluis et al., 2017; Arute et al., 2019). This limits the range of QEC codes that can be implemented directly, as many attractive codes require higher-degree connectivity or long-range couplings (Bravyi et al., 2024; Wang et al., 2026; Baspin & Williamson, 2026). Subsystem codes offer a particularly attractive way to bridge this gap. Their gauge degrees of freedom allow the same stabilizer information to be obtained from alternative sets of gauge measurements that can be lower-weight and more local (Poulin, 2005; Bombín, 2010; Bravyi, 2011). This flexibility makes subsystem codes a natural candidate for hardware-adaptive QEC on superconducting quantum processors (Chamberland et al., 2020b).

Several promising subsystem-code families illustrate this potential, including Bacon–Shor codes, subsystem surface codes, and subsystem bivariate-bicycle codes (Bacon, 2006; Bravyi et al., 2012; Liang & Chen, 2026). A common strategy in subsystem-code design is to start from a well-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

performing stabilizer or qLDPC code and construct a subsystem realization with simpler measurements (Bravyi et al., 2012; Bacon et al., 2017; Liang & Chen, 2026). However, this does not necessarily account for the physical layout of the hardware. For superconducting hardware, reducing check weight is only part of the problem: each gauge measurement must also be supported by the available couplers (Versluis et al., 2017; Chamberland et al., 2020b). Wire codes can enforce low-weight, local measurements on a target graph, but the mapping introduces additional qubit overhead that depends on the embedding (Baspin & Williamson, 2026). These considerations motivate a hardware-aware approach to code discovery, where hardware connectivity is incorporated directly into the search. We therefore ask:

Can high-quality subsystem codes be automatically discovered under different superconducting connectivity constraints?

To answer this question, we propose an LLM framework for self-evolving subsystem-code discovery under hardware constraints. Recent self-evolving and AI-scientist-style LLM approaches to QEC code design have shown that LLMs can navigate large combinatorial design spaces through iterative generation, evaluation, and refinement (Cruz-Benito et al., 2026; Yan et al., 2026; Cain et al., 2026), motivating their use for this task. Our framework follows a three-stage iterative process. First, the framework uses an LLM to generate executable construction programs for hardware-specific subsystem codes. Second, deterministic tools characterize the resulting codes, verify feasibility, and evaluate logical properties together with certified distance evidence. Third, the ranked candidates and verified diagnostic feedback are returned to the model, which uses them to refine the next round of constructions. Two features distinguish the framework. First, it enables self-evolving code discovery, in which verified code properties and failure evidence are fed back to iteratively improve subsequent generations. Second, the search is hardware constrained from the outset: candidate codes are generated directly for the target processor, so code structure, qubit placement, and realizable interactions are optimized together rather than treated as separate stages.

We evaluate the framework with GPT, DeepSeek, and Claude on four defective-chip layouts and three connectivity-design tasks based on the 105-qubit Willow processor (Google, 2025). On defective layouts, four selected GPT-generated Bacon–Shor implementations use only weight-two gauge measurements and attain circuit fault distances $d_c = 7, 7, 5, 6$, exceeding those of the native surface-code baselines. Relative to code-distance-matched routed baselines, they reduce per-cycle CZ count by 12.5%–41.2% and CZ depth by 70.0%–86.1%, without SWAP operations, while retaining $d_c = d$ under the evaluated protocols and fault models. With Willow augmentation, Claude finds a $[\![50, 2, 4, 7]\!]$ subsystem code at maximum degree 5. At maximum degree 6, GPT and DeepSeek find $[\![75, 7, 52, 5]\!]$ subsystem codes, while Claude reaches ten logical qubits with a $[\![70, 10, 0, 5]\!]$ stabilizer code, the $r = 0$ special case. With free connectivity, GPT finds a $[\![49, 1, 12, 9]\!]$ subsystem code at maximum degree 4, using 79 physical qubits and 120 couplers. Together, these results demonstrate the potential of self-evolving, verification-grounded LLM search for hardware-aware QEC design.

2 PRELIMINARIES

This section introduces the core concepts of QEC codes, subsystem codes, superconducting hardware connectivity, and automated code discovery that underlie our framework. Refer to Appendices A and B for extended preliminaries and related work.

QEC codes. QEC protects quantum information by encoding k logical qubits into n physical qubits, described by the parameters $[\![n, k, d]\!]$, where d is the code distance. Let $\mathcal{P}_n = \{\pm 1, \pm i\} \times \{I, X, Y, Z\}^{\otimes n}$ denote the n-qubit Pauli group. A stabilizer code is defined by an Abelian subgroup $\mathcal{S} = \langle S_1, \ldots, S_{n-k} \rangle \subset \mathcal{P}_n$, with $-I \notin \mathcal{S}$, whose simultaneous $+1$ eigenspace defines the code space (Gottesman, 1997). For a Pauli error $E \in \mathcal{P}_n$, its syndrome $\mathbf{s} = (s_1, \ldots, s_{n-k}) \in \mathbb{F}_2^{n-k}$ is defined by $S_i E = (-1)^{s_i} E S_i$. Thus, $s_i = 1$ indicates that E anticommutes with S_i and is detected by the corresponding stabilizer measurement. Nontrivial logical Pauli operators commute with all stabilizers but act nontrivially on the encoded information. The code distance d is the minimum weight of such an operator, and a distance-d code can correct arbitrary errors affecting up to $\lfloor (d-1)/2 \rfloor$ physical qubits.

In this work, we focus on Calderbank–Shor–Steane (CSS) codes, whose stabilizers separate into X- and Z-type generators represented by binary parity-check matrices H_X and H_Z, satisfying $H_X H_Z^\top = 0 \pmod 2$. Quantum low-density parity-check (qLDPC) codes further require the row and column weights of these matrices to remain bounded as the block length grows (Breuckmann &

Eberhardt, 2021). The row weight determines the number of qubits involved in each stabilizer check, while the column weight determines the number of checks involving each qubit. The rotated surface code is a geometrically local qLDPC code with parameters $[\![d^2, 1, d]\!]$, bounded-weight checks, and nearest-neighbor interactions on a two-dimensional lattice (Fowler et al., 2012). Its encoding rate scales as $1/n$ and its relative distance as $1/\sqrt{n}$. More general qLDPC families can achieve asymptotically good parameters with $k = \Theta(n)$ and $d = \Theta(n)$ (Panteleev & Kalachev, 2022). Bivariate-bicycle (BB) codes are a prominent qLDPC family with weight-six stabilizer checks and a degree-six Tanner graph whose interactions are not geometrically local (Bravyi et al., 2024).

Subsystem codes. Subsystem codes generalize stabilizer codes by introducing r gauge qubits whose states do not affect the protected logical information (Poulin, 2005). An $[\![n, k, r, d]\!]$ subsystem code is specified by a gauge group $\mathcal{G} \subseteq \mathcal{P}_n$ containing the global phases $\Phi = \{\pm I, \pm iI\}$. Its stabilizer group $\mathcal{S}$ is chosen as consistent Hermitian representatives of the full center $Z(\mathcal{G})/\Phi$, with $-I \notin \mathcal{S}$. The case $r = 0$ recovers stabilizer codes. Products of low-weight gauge operators can form higher-weight stabilizers, allowing stabilizer information to be recovered from simpler local measurements (Bravyi, 2011). The dressed distance is $d = \min_{P \in C_{\mathcal{P}_n}(\mathcal{S}) \setminus \mathcal{G}} \mathrm{wt}(P)$. Here $C_{\mathcal{P}_n}(\mathcal{S})$ is the set of Pauli operators commuting with every element of $\mathcal{S}$, and $\mathrm{wt}(P)$ counts the nonidentity tensor factors of P. For CSS subsystem codes, G_X and G_Z contain the proposed X- and Z-type measurement generators, including dependent rows. Their commutation matrix is $\Omega = G_X G_Z^\top$ over $\mathbb{F}_2$. If the rows of U_X and U_Z span the full coefficient kernels specified by $U_X \Omega = 0$ and $\Omega U_Z^\top = 0$, then independent row bases of $U_X G_X$ and $U_Z G_Z$ give the stabilizer matrices H_X and H_Z. This construction captures the full gauge center, including central products of measurement generators.

Superconducting quantum processors and hardware connectivity. Superconducting quantum processors can be represented by an interaction graph $H_0 = (V_0, E_0)$, where H_0 denotes the hardware interaction graph, V_0 is the set of physical qubits, and E_0 is the set of native two-qubit couplers (Fig. 1a). This graph directly constrains which multi-qubit checks can be implemented using native interactions. Surface codes are particularly well matched to this hardware model: a rotated distance-d surface code uses d^2 data qubits and $d^2 - 1$ measurement qubits arranged on a two-dimensional local lattice, with bulk weight-four stabilizers measured through nearest-neighbor interactions (Fig. 1b). For $d = 7$, this requires 49 data and 48 measurement qubits, and such a distance-seven surface-code memory has been experimentally realized on a 105-qubit superconducting processor (Google, 2025). More generally, two-dimensional geometrically local stabilizer codes obey the tradeoff $kd^2 = O(n)$, which the planar surface code saturates up to constant factors (Bravyi et al., 2010). In practical processors, however, the available interaction graph may depart from an ideal lattice because some qubits or couplers are unavailable, or because additional couplers modify the native connectivity. These hardware variations change the interactions available for syndrome extraction and therefore the set of code constructions that can be implemented directly.

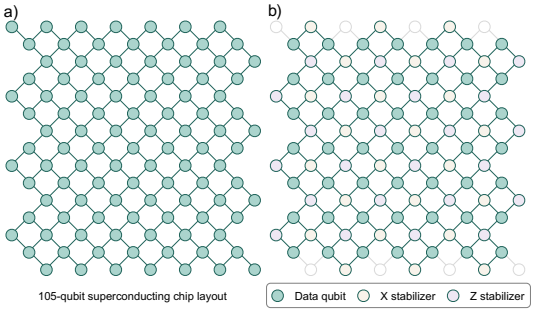

Figure 1: **Superconducting quantum processor layout. a:** An illustration of the 105-qubit Google Willow chip layout. **b:** The $[\![49, 1, 7]\!]$ rotated surface code and qubit usage.

Remark on related works. Recent LLM-based approaches have demonstrated the potential of AI-assisted QEC code discovery. IBM has used LLM-guided evolutionary search to explore bivariate-bicycle code families (Cruz-Benito et al., 2026), while LLM-assisted heuristic search has discovered lifted-product code instances for neutral-atom architectures (Cain et al., 2026). Most closely related, OmniQEC formulates QEC discovery as a self-evolving AI-scientist process that combines code generation, circuit synthesis, and decoder-based evaluation across structured qLDPC families (Yan et al., 2026). Our work instead targets self-evolving discovery of subsystem codes, where verified feedback continuously refines the search. Crucially, the search is hardware constrained from the outset: each candidate is generated directly for the target processor, including its defect pattern or connectivity budget.

3 METHOD

A central challenge in QEC discovery is to identify codes that provide useful logical protection under the constraints of a specific processor (Baspin & Krishna, 2022; Zhou et al., 2025). For superconducting devices, limited native connectivity and defective qubits or couplers restrict the available measurements (Hetényi & Wootton, 2024), while additional couplers can expand the realizable code space (Bravyi et al., 2024). These hardware restrictions interact with the algebraic structure that determines logical protection. Effective discovery therefore requires a search procedure that jointly explores code structure and physical realization, and uses evidence about constraint satisfaction to guide subsequent designs. A broader discussion of hardware-aware QEC design and superconducting connectivity is provided in Appendices A.3 and A.4.

3.1 SYSTEM ARCHITECTURE

Our framework addresses this challenge by integrating hardware-constrained code generation, deterministic verification, and objective-driven selection within an iterative discovery workflow. Following the broader paradigm of AI-scientist systems (Boiko et al., 2023; Lu et al., 2024; Yan et al., 2026), a discovery orchestrator formulates a structured search task and coordinates an LLM-based design agent with scientific tools. An overview is shown in Fig. 2. The workflow takes as input a user-defined objective, a nominal processor graph $H_0 = (V_0, E_0)$, its available subgraph $H_a = (V, E_a)$ after applying any defects, and the relevant design constraints. These constraints specify resource limits, permitted connectivity modifications, and requirements on logical protection. We instantiate this framework for subsystem codes, with qubit-role assignments and gauge generators jointly defining the candidate constructions.

Self-evolving discovery under constraints. A key difficulty in this search is that physical admissibility does not by itself ensure useful logical protection. The assignment of qubit roles determines which gauge measurements the processor can support, while the combined gauge structure determines the stabilizer center and the remaining logical degrees of freedom. Even when all proposed measurements are individually compatible with the hardware, their combination may yield $k = 0$ or a dressed distance below the required level. To address this difficulty, the framework uses verified evidence about both hardware feasibility and logical protection to guide the evolution of construction programs. Successful candidates provide reference constructions, while diagnostic information from unsuccessful proposals identifies structural limitations for subsequent revision.

At each iteration, the design agent receives the structured task together with the accumulated search context and proposes executable construction programs. Each program assigns processor physical qubits to data, measurement-ancilla, and unused roles, and then constructs the associated X- and Z-type gauge generators together with their measurement-ancilla assignments. Instantiating a program with different seeds produces related candidate realizations. The code-characterization module derives each candidate's stabilizer center, computes the numbers of logical and gauge qubits, and records certified lower bounds and constructive upper bounds on the dressed distance. A feasibility filter applies the hardware constraints and minimum protection requirements, and eligible candidates are ranked according to the user-specified objective. These evaluations and selection decisions are performed by deterministic scientific tools.

The resulting rankings and diagnostic records are returned to the orchestrator and incorporated into the discovery memory. Hardware conflicts identify unsupported interactions or resource violations, while algebraic diagnostics and verified low-weight logical operators reveal limitations of the induced code. By combining these records with promising construction programs, the orchestrator forms the search context for the next iteration, enabling the design agent to revise both qubit allocation and gauge structure. This feedback drives a self-evolving discovery process in which construction strategies are progressively refined using accumulated verification evidence. The search continues within the prescribed budget, with earlier termination when an explicitly specified stopping target is reached. The agents and tools supporting this process are described below.

3.2 CONSTRUCTION OF KEY COMPONENTS

The framework combines an LLM-based design agent with scientific tools for code characterization, feasibility assessment, and ranking. These components share a structured task specification, ensuring that the hardware constraints and protection requirements supplied to the design agent are also enforced during evaluation.

Orchestrator. The orchestrator organizes the discovery process around the processor instance, design constraints, optimization objective, and evaluation budget (Shinn et al., 2023; Yan et al., 2026).

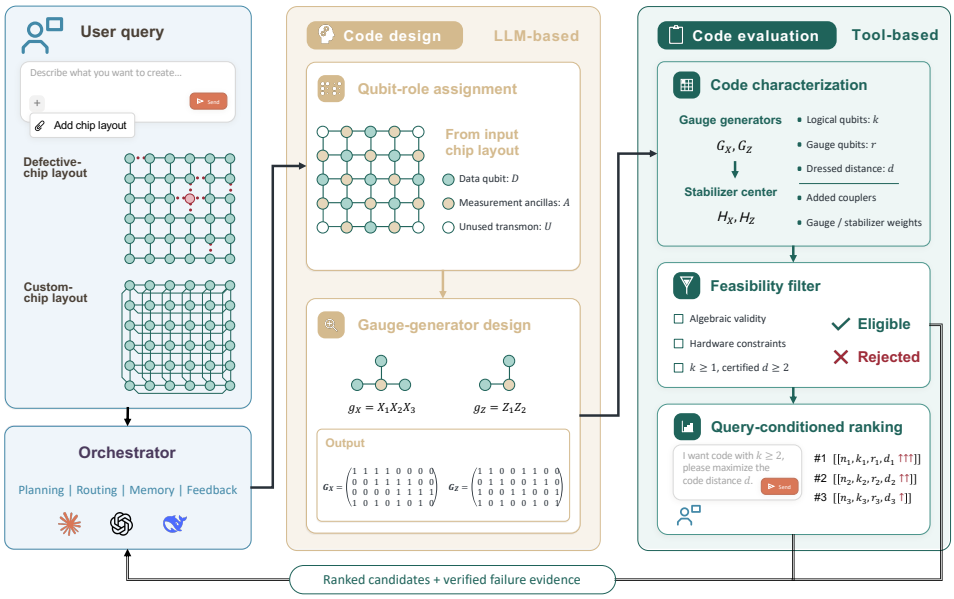

Figure 2: **System architecture for hardware-constrained QEC discovery.** The framework is instantiated here for subsystem-code design. **Left,** a user-defined objective and processor layout specify the discovery task, including defects or permitted changes to connectivity. The orchestrator coordinates planning, routing, memory, and feedback. **Middle,** the LLM-based design agent assigns physical qubits to data qubits (D), measurement ancillas (A), and unused physical qubits (U), then constructs gauge generators and their measurement-ancilla assignments. **Right,** deterministic tools derive the stabilizer center from G_X, G_Z, compute the numbers of logical and gauge qubits, establish dressed-distance bounds, and characterize hardware resources. The feasibility filter checks hardware compatibility, algebraic consistency, and baseline protection requirements; eligible candidates are ranked according to the user query. Ranked candidates and verified failure evidence return to the orchestrator through the bottom feedback path, guiding subsequent revisions of the construction programs. The illustrated layouts, gauge measurements, and matrices are schematic.

It maintains records linking each candidate to its construction program, random seed, code properties, hardware resources, and verification outcomes. For each iteration, it assembles a search context containing reference programs and summaries of previous evaluations. This context includes both promising constructions and diagnostic evidence from unsuccessful proposals, allowing subsequent designs to respond to identified hardware conflicts or limitations of logical protection. The orchestrator routes programs through scientific modules and incorporates results into the next iteration.

Code-characterization module. This module derives the subsystem structure by exact linear algebra over $\mathbb{F}_2$ (Poulin, 2005). From the submitted matrices it computes $\Omega = G_X G_Z^\top$, $r = \mathrm{rank}(\Omega)$, and $k = n - \mathrm{rank}(G_X) - \mathrm{rank}(G_Z) + r$. Independent generating matrices H_X, H_Z describe the full stabilizer center. Appendix C gives the corresponding basis construction and dimension checks. Canonical gauge row spaces identify algebraically identical candidates and permit distance-evidence reuse when the labelled data set is unchanged. Hardware resources, gauge weights, and stabilizer weights are computed from the submitted measurement generators and their ancilla assignments.

Distance evaluation records a lower bound $\underline{d}$ and a constructive upper bound $\overline{d}$, with $\underline{d} \leq d \leq \overline{d}$. Completed exclusion of nontrivial dressed logical operators through weight w establishes $\underline{d} \geq w+1$; an independently checked logical operator supplies an upper bound through its weight. Each witness must commute with the full stabilizer center and lie outside the gauge group. MILP-based evaluation records an exact distance when the solver reports optimality and the returned witness is independently revalidated. Matching established bounds also determine an exact distance. Unfinished verification retains only the evidence already obtained (refer to Appendix D for more details).

Feasibility filter. Hardware checks cover the site partition, measurement supports, ancilla assignments, and required couplers. For a candidate C, let $F(C)$ denote the set of direct couplers connecting each measurement ancilla to the data qubits in its measurement support. For fixed connectivity, $F(C) \subseteq E_a$ is required. In Willow augmentation, all native couplers are retained, so the installed

set is $E = E_0 \cup F(C)$ and the added set is $E \setminus E_0$. In free-connectivity design, the installed graph uses the required set $E = F(C)$ on the available physical qubits. Maximum degree is computed from the full installed graph, including retained native couplers. Measurement-weight and resource limits follow the task specification. Baseline protection requires $k \geq 1$ and $\underline{d} \geq 2$. Rejection records distinguish violated requirements from insufficient protection evidence.

Query-conditioned ranking and feedback. The ranker orders eligible realizations using the objective and metric priorities specified by the task. Query-specific target requirements are assessed separately from baseline eligibility. For example, a task that maximizes distance subject to $k \geq k_{\min}$ first identifies candidates meeting the logical-capacity requirement and compares their certified distance lower bounds. Each ranked entry remains linked to its construction program and seed. Feedback summarizes outcomes across the realizations of each program and records parent–child changes in qubit roles, gauge supports, hardware interactions, and algebraic structure. Candidates that miss a target or fail verification can still contribute diagnostic evidence. These records are returned alongside the ranking, providing the orchestrator with both reference constructions and concrete constraints to inform subsequent program revisions. Appendix E specifies the program interface, caching, reference selection, memory, and recovery protocol.

4 Numerical results

We evaluate our framework in two complementary settings that couple code design to superconducting hardware constraints. On defective chip layouts, the objective is to recover logical protection using the surviving connectivity. On custom chip layouts, we explore how connectivity changes within a 105-physical-qubit budget can improve the protection of multiple logical qubits, increase encoding capacity, or reduce the connectivity required for a target distance. We instantiate the framework with GPT, DeepSeek, and Claude, and assess the resulting codes through their verified code parameters and implementation resources.

4.1 Code discovery on defective chip layouts

Hardware defects can disable interactions required by a code's measurement structure, making an otherwise suitable construction incompatible with the processor (Auger et al., 2017; Drouet et al., 2026). We therefore test whether jointly adapting qubit roles and gauge generators can sustain logical protection using only the surviving hardware graph.

Settings. We use the Willow topology with 105 superconducting transmon qubits and 182 couplers (Google, 2025), and construct four fixed synthetic defect masks informed by Google Cirq calibration records: 2, 4, or 9 unavailable couplers, and 2 unavailable physical qubits together with 2 additional unavailable couplers. High CZ error rates and short T_1 values identify candidate couplers and qubits for removal, respectively. Each mask preserves graph connectivity and introduces no new articulation vertex. For each mask, the search maximizes dressed distance on the surviving graph, without adding couplers. Calibration values and defect masks are provided in Appendix F.1.

Baselines and metrics. We compare against two rotated surface-code implementations on the same surviving graph. The *native* baseline selects the largest directly embeddable template over rotations, reflections, and translations. The *routed* baseline matches the best discovered code distance, implements blocked interactions through round-trip SWAP routing, and reassigns roles that fall on unavailable qubits. These baselines cover direct template embedding and SWAP-based adaptation; specialized check modifications, gauge-based superchecks, and ancilla repurposing offer complementary approaches (Auger et al., 2017; Leroux et al., 2025). Their relation to our defect model is discussed in Appendix F.3.

In addition to dressed distance d, we evaluate the returned implementations through circuit fault distance d_c, defined as the minimum number of elementary faults that cause an undetected protected logical error. We report the smaller d_c between X- and Z-basis memories with d measurement cycles; fault definitions and certificates appear in Appendix F.4. Under a common compilation and scheduling protocol, we record the number of occupied physical qubits Q, including data qubits, measurement ancillas, and routing qubits, and the maximum measurement weight w. We also report CZ count, CZ depth, and SWAPs per cycle, with SWAP decompositions included in the CZ count.

Results. Fig. 3 shows four GPT-generated subsystem implementations that attain the best dressed distances found for the corresponding masks. Their distances are $d = 7, 7, 5, 6$ for the 2-, 4-, and 9-coupler masks and the mixed qubit–coupler mask, respectively, compared with $d = 5, 4, 3, 4$ for the native surface-code baseline (Tab. 1). All four are hardware-specific realizations of square Bacon–

Table 1: Code parameters and circuit resources on defective chip layouts. In defect labels, C/Q denote unavailable couplers/qubits; Q counts occupied physical qubits and w is the maximum measurement weight. CZ count includes gates used to implement SWAP operations, with nonzero SWAP counts shown in parentheses.

Defects	Method	$[\![n, k, r, d]\!]$	d_c	Q	w	CZ count	CZ depth
2C	Ours (GPT)	$[\![49, 1, 36, 7]\!]$	7	93	2	168	12
	Ours (DeepSeek)	$[\![49, 1, 36, 7]\!]$	7	94	2	168	12
	Ours (Claude)	$[\![49, 1, 36, 7]\!]$	7	102	2	168	12
	Surface (native)	$[\![25, 1, 0, 5]\!]$	5	49	4	80	16
	Surface (routed)	$[\![49, 1, 0, 7]\!]$	6	97	4	192 (8)	40
4C	Ours (GPT)	$[\![49, 1, 36, 7]\!]$	7	99	2	168	12
	Ours (DeepSeek)	$[\![49, 1, 36, 7]\!]$	7	92	2	168	12
	Ours (Claude)	$[\![49, 1, 36, 7]\!]$	7	84	2	168	10
	Surface (native)	$[\![16, 1, 0, 4]\!]$	4	31	4	48	16
	Surface (routed)	$[\![49, 1, 0, 7]\!]$	6	97	4	216 (16)	52
9C	Ours (GPT)	$[\![25, 1, 16, 5]\!]$	5	47	2	80	10
	Ours (DeepSeek)	$[\![28, 1, 18, 4]\!]$	4	51	2	90	12
	Ours (Claude)	$[\![36, 4, 0, 3]\!]$	2	58	4	96	16
	Surface (native)	$[\![9, 1, 0, 3]\!]$	3	17	4	24	16
	Surface (routed)	$[\![25, 1, 0, 5]\!]$	4	50	4	128 (16)	64
2Q + 2C	Ours (GPT)	$[\![36, 1, 25, 6]\!]$	6	68	2	120	10
	Ours (DeepSeek)	$[\![36, 1, 25, 6]\!]$	6	71	2	120	10
	Ours (Claude)	$[\![36, 1, 25, 6]\!]$	6	68	2	120	12
	Surface (native)	$[\![16, 1, 0, 4]\!]$	4	31	4	48	16
	Surface (routed)	$[\![36, 1, 0, 6]\!]$	4	72	4	204 (28)	72

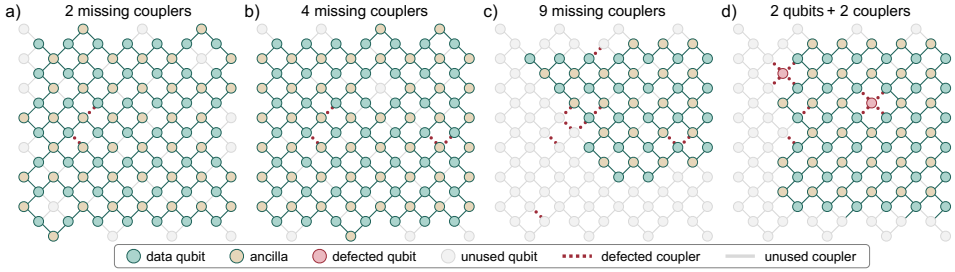

Figure 3: **Subsystem codes discovered with the GPT backend.** The panels correspond to the four defect masks in Tab. 1, respectively. All four use weight-two gauge measurements on the surviving connectivity and retain $d_c = d$.

Shor codes (Bacon, 2006), using only weight-two gauge measurements supported by the surviving graph, indicating that LLMs utilize their internal knowledge to fit the hardware constraints.

These implementations also retain $d_c = d$. Surface codes routed to match their code distances attain $d_c = 6, 6, 4, 4$ under the same evaluation protocol. Relative to those routed implementations, the discovered implementations reduce per-cycle CZ count by 12.5%–41.2% and CZ depth by 70.0%–86.1%, and require no SWAPs. The comparison therefore shows gains in circuit fault distance alongside lower gate and scheduling overhead.

4.2 Code discovery on custom chip layouts

Conventional planar surface-code and triangular color-code patches each encode a single logical qubit (Google, 2025; Chamberland et al., 2020a). Protecting multiple logical qubits with separate patches therefore divides a fixed processor budget among their respective data qubits and measurement ancillas. We further investigate whether tailored connectivity can support multiple logical qubits with useful protection within the same hardware budget. Superconducting architectures with crossing and long-range couplers provide access to richer measurement patterns (Wang et al., 2026), while frequency-crowding considerations motivate limiting the number of couplers incident on each qubit. These considerations lead to three complementary objectives: improving protection, increasing logical capacity, and reducing the connectivity required for a prescribed distance.

Settings. All three tasks use at most 105 physical qubits, including data qubits and measurement ancillas. The first two augment the Willow layout while retaining all 182 native couplers, and target the protection of multiple logical qubits ($k \geq 2$). The third permits the coupler graph to be designed anew. Tab. 2 summarizes the objectives and constraints, with Δ denoting the maximum installed qubit degree.

Table 2: Custom chip layout tasks with $Q \leq 105$.

Task	Objective	Constraints
Protection	$\max d$	$k \geq 2,\ \Delta \leq 5$
Capacity	$\max k$	$k \geq 2,\ \Delta \leq 6$
Connectivity	$\min \Delta$	$d \geq 9$

Table 3: Code parameters and hardware resources for the three tasks in Tab. 2.

Task	Method	$[\![n, k, r, d]\!]$	Q	$\lvert E \setminus E_0 \rvert$	$\lvert E \rvert$	Δ	w
Protection ($\Delta \leq 5$)	Ours (GPT)	$[\![54, 2, 38, 6]\!]$	99	24	206	5	2
	Ours (DeepSeek)	$[\![32, 2, 18, 4]\!]$	50	0	182	4	2
	Ours (Claude)	$[\![50, 2, 4, 7]\!]$	98	26	208	5	4
	TB baseline	$[\![30, 4, 0, 5]\!]$	56	–	130	5	5
Capacity ($\Delta \leq 6$)	Ours (GPT)	$[\![75, 7, 52, 5]\!]$	105	100	282	6	2
	Ours (DeepSeek)	$[\![75, 7, 52, 5]\!]$	105	101	283	6	2
	Ours (Claude)	$[\![70, 10, 0, 5]\!]$	100	66	248	6	6
	BB baseline	$[\![54, 8, 0, 6]\!]$	100	–	276	6	6
Connectivity ($d \geq 9$)	Ours (GPT)	$[\![49, 1, 12, 9]\!]$	79	–	120	4	4
	Ours (DeepSeek)	$[\![63, 1, 4, 9]\!]$	102	–	180	6	6
	Ours (Claude)	$[\![61, 1, 0, 9]\!]$	91	–	156	6	6
	Color baseline	$[\![61, 1, 0, 9]\!]$	91	–	156	6	6

TB/BB use one ancilla per independent original check; the color-code baseline reuses each face ancilla for its X- and Z-type checks.

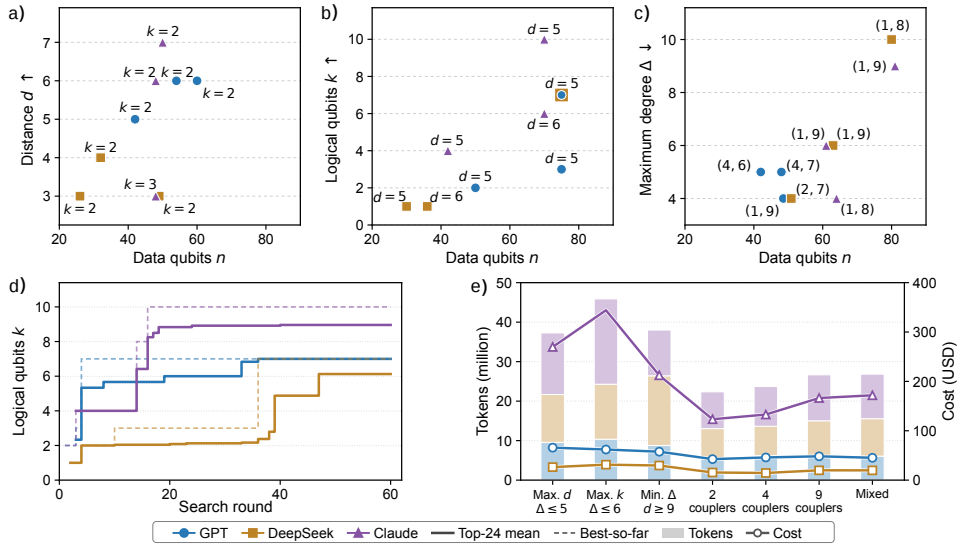

Figure 4: **Code discovery and search resources. a–c**: Representative candidates found across search rounds for the protection, capacity, and connectivity tasks, respectively. Annotations show k in (a), d in (b), and (k, d) in (c). **d**: Best k (dashed) and top-24 mean k (solid) during the capacity search. **e**: Token usage (bars, left axis) and API cost (lines, right axis) across all settings.

The protection task asks how far distance can be increased while encoding multiple logical qubits under a restricted degree budget. The capacity task instead uses the additional connectivity to increase k. The connectivity task reverses the design question by minimizing Δ at a prescribed distance $d \geq 9$. This target exceeds the native Willow bound $d \leq 7$ established in Appendix A.4 under our direct-measurement model, with fixed data/ancilla roles and one native ancilla star per measured generator. It therefore tests whether redesigning the connection pattern can support higher distance while keeping the required degree low.

Baselines and metrics. We evaluate code distance and logical capacity together with occupied physical qubits Q, maximum measurement weight w, maximum installed degree Δ, total couplers $|E|$, and added couplers $|E \setminus E_0|$, where E_0 and E denote the native and installed edge sets. For the augmentation tasks, both degree and coupler counts include all retained native couplers.

We use sparse-check codes as references for the three objectives. The $[\![30, 4, 0, 5]\!]$ trivariate-bicycle (TB) code (Voss et al., 2025) provides a multi-logical-qubit reference at $\Delta \leq 5$. The $[\![54, 8, 0, 6]\!]$ BB code (Wang & Mueller, 2026) provides a capacity reference at $\Delta \leq 6$ and occupies $Q = 100$ physical qubits. Their tabulated connectivity counts describe standalone measurement graphs. For these fixed TB and BB measurement constructions, Appendix F.2 shows our bounded embedding searches found no layout retaining all 182 native Willow couplers while satisfying $\Delta \leq 5$ and $\Delta \leq 6$. Our reported augmentation designs, in contrast, include verified layouts satisfying native-edge retention and the corresponding degree caps. For the connectivity task, the triangular $[\![61, 1, 0, 9]\!]$ color code (Chamberland et al., 2020a) supplies a reference with the same $k = 1, d = 9$ as the reported discoveries. Ancilla-allocation conventions are specified in Tab. 3.

Results. Tab. 3 reports the resulting designs, and Fig. 4a–c presents representative candidates for the three objectives. **For the protection task**, GPT discovers a $[\![54, 2, 38, 6]\!]$ subsystem implementation with 24 added couplers and weight-two gauge measurements. Claude finds a $[\![50, 2, 4, 7]\!]$ design with 26 added couplers. These constructions preserve the native connectivity while protecting two logical qubits at distances six and seven, respectively. Appendix G.3 gives the gauge matrices of the GPT protection-task example, and Appendix Tab. 7 lists additional code-parameter profiles across all seven settings. **For the capacity task,** GPT and DeepSeek both find $[\![75, 7, 52, 5]\!]$ subsystem codes using weight-two measurements within the $Q = 105$ budget. Claude's $[\![70, 10, 0, 5]\!]$ design encodes ten logical qubits using $Q = 100$. At the same physical-qubit count, the BB reference encodes eight logical qubits at $d = 6$, whereas the discovered design has $d = 5$ and uses 248 rather than 276 couplers. The results thus expose concrete choices between logical capacity, distance, and measurement weight within the available hardware budget. **For the connectivity task,** GPT identifies a $[\![49, 1, 12, 9]\!]$ subsystem code with $\Delta = 4$, compared with $\Delta = 6$ for the color-code reference at the same $k = 1$ and $d = 9$. It also reduces the physical-qubit count from 91 to 79, the coupler count from 156 to 120, and the maximum measurement weight from six to four. This comparison illustrates how joint code and connectivity design can reduce hardware requirements and measurement weight while maintaining the same logical capacity and code distance.

Two clear patterns emerge. First, the benefit of additional connectivity depends strongly on how it is used: modest augmentation improves distance, while greater freedom can increase logical capacity or reduce the connectivity required for high-distance codes. Second, the three models show different search behaviors. DeepSeek is less consistent in reaching the strongest solutions, whereas Claude often converges to strong but more conventional stabilizer constructions, suggesting greater reliance on familiar code structures. GPT shows the most balanced performance, consistently finding competitive subsystem codes with favorable protection, measurement weight, and hardware cost.

Search dynamics and cost. Fig. 4d tracks the capacity search through both the best logical capacity found and the mean k of the top 24 distinct archived metric profiles. Improvements in both quantities show that successive rounds enrich the pool of competitive candidates alongside improving the best result. The corresponding trajectories for the other two objectives appear in Appendix Fig. 6. Fig. 4e reports token consumption and API costs across all seven settings. Model configurations, continuation protocols, and resource accounting are given in Appendix F.2.

5 Discussion

Our results on defective and custom chip layouts demonstrate the potential of LLM-assisted subsystem-code discovery under hardware constraints. QEC offers a natural setting for such discovery: its vast search space makes exhaustive enumeration impractical, while its algebraic, geometric, and hardware structure provides regularities that generative models can exploit. Coupling hardware-conditioned generation with deterministic verification turns feedback on code properties and physical feasibility into guidance for successive revisions.

Effective search must also reflect the representation of the target code family. Many structured stabilizer-code constructions enforce algebraic validity by design. In subsystem-code search, however, qubit assignments and gauge choices can eliminate logical qubits, reduce distance, or require hardware-infeasible measurements. These failure modes make the integration of generation, code characterization, feasibility checks, and diagnostic feedback central to successful search.

Limitations. We report preliminary logical-error-rate results in Appendix G.2, but a comprehensive circuit-level evaluation of the discovered codes remains future work. Extending this evaluation across diverse gauge constructions requires further development of subsystem-code circuit synthesis and decoding. Syndrome extraction must balance parallelism, ancilla scheduling, and fault propagation to limit hook-induced distance reduction; otherwise, circuit quality can obscure the merits of the code itself. Assessing the benefits of low-weight gauge measurements would also benefit from noise models that more closely reflect device-specific coherence, residual coupling, and crosstalk. Advances in these tools would support more faithful evaluation and richer feedback for joint code, circuit, and hardware design.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript preparation and software development, including language polishing, refinement of technical exposition, and assistance with code implementation and revision. In addition, the proposed agentic system uses ChatGPT, Claude, and DeepSeek as external LLM services within its search workflow. The authors are responsible for ensuring the correctness of AI-assisted code and the accuracy of AI-assisted text, and take full responsibility for the methods, software, results, and conclusions presented in this work.

ETHICS STATEMENT

This work develops an agentic system for discovering subsystem quantum error-correcting codes. The research concerns mathematical code constructions and computational search procedures, with the aim of supporting advances in quantum error correction and fault-tolerant quantum computation. It does not involve human participants or personal data. Within the scope of the system and research activities presented, we do not identify specific ethical concerns or anticipate direct adverse impacts on individuals or society.

REPRODUCIBILITY STATEMENT

During peer review, the source code and supporting research materials will be made available to reviewers upon reasonable request through channels that preserve submission anonymity. Upon acceptance, we will publicly release the source code of the agentic system, search configurations, prompts, evaluation and verification scripts, and the subsystem code constructions underlying the reported results. The accompanying documentation will specify the external LLM model identifiers, inference settings, and software dependencies, together with instructions for running the system and verifying the reported code properties. Reproducing experiments involving LLM calls requires access to the corresponding external services.

REFERENCES

Panos Aliferis and Andrew W Cross. Subsystem fault tolerance with the bacon-shor code. *Physical review letters*, 98(22):220502, 2007.

Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *nature*, 574(7779):505–510, 2019.

James M Auger, Hussain Anwar, Mercedes Gimeno-Segovia, Thomas M Stace, and Dan E Browne. Fault-tolerance thresholds for the surface code with fabrication errors. *Physical Review A*, 96(4): 042316, 2017.

Dave Bacon. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Physical Review A—Atomic, Molecular, and Optical Physics*, 73(1):012340, 2006.

Dave Bacon, Steven T Flammia, Aram W Harrow, and Jonathan Shi. Sparse quantum codes from quantum circuits. *IEEE Transactions on Information Theory*, 63(4):2464–2479, 2017.

Nouédyn Baspin and Anirudh Krishna. Connectivity constrains quantum codes. *Quantum*, 6:711, 2022.

Nouédyn Baspin and Dominic Williamson. Wire codes. *Quantum*, 10:2083, 2026.

Dolev Bluvstein et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626 (7997):58–65, 2024. doi: 10.1038/s41586-023-06927-3.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Héctor Bombín. Topological subsystem codes. *Physical Review A—Atomic, Molecular, and Optical Physics*, 81(3):032301, 2010.

Sergey Bravyi. Subsystem codes with spatially local generators. *Physical Review A—Atomic, Molecular, and Optical Physics*, 83(1):012320, 2011.

Sergey Bravyi, David Poulin, and Barbara Terhal. Tradeoffs for reliable quantum information storage in 2d systems. *Physical review letters*, 104(5):050503, 2010.

Sergey Bravyi, Guillaume Duclos-Cianci, David Poulin, and Martin Suchara. Subsystem surface codes with three-qubit check operators. *arXiv preprint arXiv:1207.1443*, 2012.

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(8005): 778–782, 2024.

Nikolas P Breuckmann and Jens Niklas Eberhardt. Quantum low-density parity-check codes. *PRX quantum*, 2(4):040101, 2021.

Madelyn Cain, Qian Xu, Robbie King, Lewis RB Picard, Harry Levine, Manuel Endres, John Preskill, Hsin-Yuan Huang, and Dolev Bluvstein. Shor's algorithm is possible with as few as 10,000 reconfigurable atomic qubits. *arXiv preprint arXiv:2603.28627*, 2026.

Earl Campbell. A series of fast-paced advances in quantum error correction. *Nature Reviews Physics*, 6(3):160–161, 2024.

Chenfeng Cao, Chao Zhang, Zipeng Wu, Markus Grassl, and Bei Zeng. Quantum variational learning for quantum error-correcting codes. *Quantum*, 6:828, 2022.

Christopher Chamberland, Aleksander Kubica, Theodore J Yoder, and Guanyu Zhu. Triangular color codes on trivalent graphs with flag qubits. *New Journal of Physics*, 22(2):023019, 2020a.

Christopher Chamberland, Guanyu Zhu, Theodore J Yoder, Jared B Hertzberg, and Andrew W Cross. Topological and subsystem codes on low-degree graphs with flag qubits. *Physical Review X*, 10(1):011022, 2020b.

Gregory M Crosswhite and Dave Bacon. Automated searching for quantum subsystem codes. *Physical Review A—Atomic, Molecular, and Optical Physics*, 83(2):022307, 2011.

Juan Cruz-Benito, Andrew W Cross, David Kremer, and Ismael Faro. Evolutionary discovery of bivariate bicycle codes with llm-guided search. *arXiv preprint arXiv:2606.02418*, 2026.

Nathalie P De Leon, Kohei M Itoh, Dohun Kim, Karan K Mehta, Tracy E Northup, Hanhee Paik, BS Palmer, Nitin Samarth, Sorawis Sangtawesin, and David W Steuerman. Materials challenges and opportunities for quantum computing hardware. *Science*, 372(6539):eabb2823, 2021.

Julien M Drouet, Xanda C Kolesnikow, Campbell K McLauchlan, Georgia M Nixon, Seok-Hyung Lee, Dominic J Williamson, Stephen D Bartlett, Benjamin J Brown, and Robin Harper. Error correction on an array of superconducting qubits with defective components. *arXiv preprint arXiv:2607.12118*, 2026.

Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A—Atomic, Molecular, and Optical Physics*, 86(3):032324, 2012.

Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, 2021. URL `https://github.com/quantumlib/Stim`.

Google. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949): 676–681, 2023.

Google. Quantum error correction below the surface code threshold. *Nature*, 638(8052):920–926, 2025.

Google Quantum AI. Calibration metrics. Cirq documentation, 2026a. URL https://quantumai.google/cirq/google/calibration. Last updated July 23, 2026. Accessed September 25, 2026.

Google Quantum AI. Google devices. Cirq documentation, 2026b. URL https://quantumai.google/cirq/google/devices. Last updated September 22, 2026. Accessed September 25, 2026.

Daniel Gottesman. *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

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

Bence Hetényi and James R Wootton. Creating entangled logical qubits in the heavy-hex lattice with topological codes. *Prx Quantum*, 5(4):040334, 2024.

David Kribs, Raymond Laflamme, and David Poulin. Unified and generalized approach to quantum error correction. *Physical review letters*, 94(18):180501, 2005.

Catherine Leroux, Sophia F Lin, Przemyslaw Bienias, Krishanu R Sankar, Asmae Benhemou, Aleksander Kubica, and Joseph K Iverson. Snakes and ladders: Adapting the surface code to defects. *PRX Quantum*, 6(4):040302, 2025.

Zijian Liang and Yu-An Chen. Topological subsystem bivariate bicycle codes with four-qubit check operators. *arXiv preprint arXiv:2605.04151*, 2026.

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

Zidu Liu and Florian Marquardt. Large-language-model discovery of quantum ldpc codes through structured concept evolution. *arXiv preprint arXiv:2606.24808*, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Caroline Mauron, Terry Farrelly, and Thomas M Stace. Optimization of tensor network codes with reinforcement learning. *New Journal of Physics*, 26(2):023024, 2024.

Hendrik Poulsen Nautrup, Nicolas Delfosse, Vedran Dunjko, Hans J Briegel, and Nicolai Friis. Optimizing quantum error correction codes with reinforcement learning. *Quantum*, 3:215, 2019.

Jan Olle, Remmy Zen, Matteo Puviani, and Florian Marquardt. Simultaneous discovery of quantum error correction codes and encoders with a noise-aware reinforcement learning agent. *npj Quantum Information*, 10(1):126, 2024.

Pavel Panteleev and Gleb Kalachev. Quantum ldpc codes with almost linear minimum distance. *IEEE Transactions on Information Theory*, 68(1):213–229, 2021.

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*, pp. 375–388, 2022.

David Poulin. Stabilizer formalism for operator quantum error correction. *Physical review letters*, 95(23):230504, 2005.

John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

Dongheng Qian and Tianyi Li. Multi-agent discovery of practical quantum ldpc codes. *arXiv preprint arXiv:2608.08996*, 2026.

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.

Omer S Sella, Robert Pinsler, and Thomas Heinis. Designing quantum error correcting codes to fit decoders via reinforcement learning. *arXiv preprint arXiv:2608.15754*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Vincent Paul Su, ChunJun Cao, Hong-Ye Hu, Yariv Yanay, Charles Tahan, and Brian Swingle. Discovery of optimal quantum codes via reinforcement learning. *Physical review applied*, 23(3): 034048, 2025.

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.

Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. *Physical Review A*, 90(6):062320, 2014.

Richard Versluis, Stefano Poletto, Nader Khammassi, Brian Tarasinski, Nadia Haider, David J Michalak, Alessandro Bruno, Koen Bertels, and Leonardo DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *Physical Review Applied*, 8(3):034021, 2017.

Lukas Voss, Sim Jian Xian, Tobias Haug, and Kishor Bharti. Multivariate bicycle codes. *Physical Review A*, 111(6):L060401, 2025.

Ke Wang, Zhide Lu, Chuanyu Zhang, Gongyu Liu, Jiachen Chen, Yanzhe Wang, Yaozu Wu, Shibo Xu, Xuhao Zhu, Feitong Jin, et al. Demonstration of low-overhead quantum error correction codes. *Nature Physics*, 22(2):308–314, 2026.

Ming Wang and Frank Mueller. Coprime bivariate bicycle codes and their layouts on cold atoms. *Quantum*, 10:2009, 2026.

Mark A Webster and Dan E Browne. Engineering quantum error correction codes using evolutionary algorithms. *IEEE Transactions on Quantum Engineering*, 6:1–14, 2025.

Ge Yan, Shanchuan Li, Pengyue Ma, Qixin Zhang, Pingchuan Ma, Jianping Wang, Min-Hsiu Hsieh, and Yuxuan Du. Omniqec: discovering practical quantum error-correcting codes by an ai scientist. *arXiv preprint arXiv:2607.25865*, 2026.

Hengyun Zhou, Madelyn Cain, and Mikhail D Lukin. Opportunities in full-stack design of low-overhead fault-tolerant quantum computation. *Nature Computational Science*, 5(12):1110–1119, 2025.

A EXTENDED PRELIMINARIES

A.1 QUANTUM ERROR CORRECTION (QEC) CODES

Stabilizer codes. Consider a system of n physical qubits with Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$, and let $\mathcal{P}_n = \{\pm 1, \pm i\} \times \{I, X, Y, Z\}^{\otimes n}$ denote the n-qubit Pauli group. An $[\![n, k, d]\!]$ stabilizer code is specified by an Abelian subgroup $\mathcal{S} \subset \mathcal{P}_n$ such that $-I \notin \mathcal{S}$. Its code space $\mathcal{C}$ is the simultaneous $+1$ eigenspace of all operators in $\mathcal{S}$. If the code encodes k logical qubits, then $\mathcal{S}$ has $n - k$ independent generators: $\mathcal{S} = \langle S_1, \ldots, S_{n-k} \rangle$.

For a Pauli error $E \in \mathcal{P}_n$, its syndrome $\sigma(E) = (s_1, \ldots, s_{n-k}) \in \mathbb{F}_2^{n-k}$ is defined by $S_i E = (-1)^{s_i} E S_i$. A nonzero syndrome indicates that E anticommutes with at least one stabilizer generator and is therefore detected. Two Pauli errors E and F have the same syndrome if and only if $E^\dagger F$ commutes with every element of $\mathcal{S}$.

A Pauli operator preserves the code space if it commutes with every element of $\mathcal{S}$. The set of all such operators is the centralizer $C_{\mathcal{P}_n}(\mathcal{S})$. Since global phases have no physical effect and stabilizers act trivially on the code space, nontrivial logical Pauli operators correspond to nonidentity elements of $C_{\mathcal{P}_n}(\mathcal{S})/(\Phi\mathcal{S})$, where $\Phi = \{\pm I, \pm iI\}$. The code distance is therefore

$$d = \min_{P \in C_{\mathcal{P}_n}(\mathcal{S}) \setminus \Phi\mathcal{S}} \mathrm{wt}(P). \tag{1}$$

Consequently, the code can correct any error supported on at most $\lfloor (d-1)/2 \rfloor$ physical qubits.

Calderbank–Shor–Steane (CSS) and quantum low-density parity-check (qLDPC) codes. In this work, we focus on CSS codes, whose stabilizer generators can be chosen to be either purely X-type or purely Z-type. These two sets of generators are represented by binary parity-check matrices $H_X \in \mathbb{F}_2^{m_X \times n}$ and $H_Z \in \mathbb{F}_2^{m_Z \times n}$, where each row specifies the support of a stabilizer generator. The mutual commutation of the two types of generators is equivalent to the orthogonality condition

$$H_X H_Z^\top = 0 \pmod 2. \tag{2}$$

If the X- and Z-components of a Pauli error are represented by column vectors $e_X, e_Z \in \mathbb{F}_2^n$, then the Z-type checks detect the X-component and the X-type checks detect the Z-component. The corresponding syndromes are therefore $s_Z = H_Z e_X$ and $s_X = H_X e_Z$.

Within the CSS framework, qLDPC codes are distinguished by the sparsity of their parity-check matrices (Tillich & Zémor, 2014; Panteleev & Kalachev, 2022). For a binary matrix H, let $w_r(H)$ and $w_c(H)$ denote its maximum row and column weights, respectively. A family of CSS codes is qLDPC if there exist constants w and q, independent of the block length n, such that

$$\max\{w_r(H_X), w_r(H_Z)\} \le w, \qquad \max\{w_c(H_X), w_c(H_Z)\} \le q. \tag{3}$$

These two bounds have a direct physical interpretation. The row-weight bound limits the number of data qubits involved in each stabilizer check, while the column-weight bound limits the number of checks involving each data qubit. Thus, both the check weight and the qubit degree remain constant as the code size increases.

The qLDPC framework encompasses a broad range of constructions. Hypergraph-product (HGP) codes are built from pairs of classical LDPC codes and provide a canonical route to finite-rate qLDPC families (Tillich & Zémor, 2014). Lifted-product codes generalize this product-based approach by replacing entries of small base matrices with sparse lifted blocks, thereby expanding the available qLDPC design space (Panteleev & Kalachev, 2021). A related class is formed by two-block group-algebra (2bGA) codes, whose parity-check matrices are constructed from two elements of a finite group algebra (Lin & Pryadko, 2024). Bivariate-bicycle (BB) codes are a prominent abelian instance of this construction (Bravyi et al., 2024).

The surface code is another important qLDPC family and is distinguished by its geometric locality. A square rotated planar surface code of distance d has parameters $[\![d^2, 1, d]\!]$, with weight-four stabilizers in the bulk and lower-weight checks along the boundaries. Both the stabilizer weight w and the number q of checks involving each data qubit are bounded independently of d, with $w, q \le 4$. In addition, each stabilizer acts only on nearby qubits arranged on a two-dimensional square lattice. This geometric locality allows the code to be implemented using nearest-neighbor interactions. Together, its low check weight, bounded qubit degree, and local connectivity make the surface code well suited to superconducting quantum processors, as discussed in Appendix A.3.

Asymptotic code performance. A central objective in QEC is to reduce the physical-qubit overhead required to protect a growing amount of logical information. For a family of $[\![n, k_n, d_n]\!]$ codes, two standard measures of asymptotic performance are the encoding rate $R_n = k_n/n$ and the relative distance $\delta_n = d_n/n$. A code family is asymptotically good if both quantities remain bounded away from zero as n increases:

$$\liminf_{n\to\infty} R_n > 0, \qquad \liminf_{n\to\infty} \delta_n > 0. \tag{4}$$

The surface code does not satisfy this condition. For a square rotated surface code with $n = d^2$ and $k = 1$, the encoding rate and relative distance are

$$R = \frac{1}{d^2} = \frac{1}{n}, \qquad \delta = \frac{1}{d} = \frac{1}{\sqrt{n}}. \tag{5}$$

Both quantities therefore vanish as the code size increases. In contrast, suitable HGP code families achieve $k = \Theta(n)$ and $d = \Theta(\sqrt{n})$, providing a constant encoding rate but a vanishing relative distance. More recent qLDPC constructions have improved the minimum-distance scaling from nearly linear to linear in n, culminating in asymptotically good code families (Panteleev & Kalachev, 2021; 2022).

These improvements motivate the exploration of qLDPC codes beyond the surface code when a higher encoding rate or stronger distance scaling is needed under a fixed physical-qubit budget. However, many practical qLDPC constructions involve larger constant check weights or qubit degrees than the surface code and require interactions between qubits that are not geometrically adjacent. Their improved code parameters must therefore be considered together with the additional complexity of hardware implementation.

A.2 Subsystem codes

Subsystem-code formalism. Subsystem quantum error-correcting codes generalize stabilizer codes by allowing part of the encoded Hilbert space to carry gauge degrees of freedom that are irrelevant to the protected logical information. Let $\Phi = \{\pm I, \pm iI\}$ denote the subgroup of global Pauli phases. A subsystem code is specified by a generally non-Abelian gauge group $\mathcal{G} \subseteq \mathcal{P}_n$ containing Φ. The stabilizer group is obtained from the center of $\mathcal{G}$, where

$$Z(\mathcal{G}) = \mathcal{G} \cap C_{\mathcal{P}_n}(\mathcal{G}), \qquad \mathcal{S} \simeq Z(\mathcal{G})/\Phi. \tag{6}$$

After appropriate Hermitian representatives and signs are chosen, the simultaneous $+1$ eigenspace of $\mathcal{S}$ decomposes as $\mathcal{C} \simeq \mathcal{H}_L \otimes \mathcal{H}_G$, where $\mathcal{H}_L$ stores k protected logical qubits and $\mathcal{H}_G$ contains r gauge qubits. Such a code is conventionally denoted by $[\![n, k, r, d]\!]$. Gauge operators may act nontrivially on $\mathcal{H}_G$, but they act trivially on the protected logical subsystem. Consequently, physical errors that differ by an element of $\mathcal{G}$ are equivalent with respect to the logical information, extending the stabilizer equivalence discussed above.

The distinction between stabilizer and gauge generators is central to subsystem codes. In an ordinary stabilizer code, all check operators belong to the mutually commuting stabilizer group $\mathcal{S}$. A subsystem code instead allows a larger set of gauge generators that need not commute with one another, while the stabilizers arise as commuting products of these generators. A canonical generating set of the gauge group can be written schematically as

$$\mathcal{G} = \langle \Phi, \mathcal{S}, X_{g,1}, Z_{g,1}, \ldots, X_{g,r}, Z_{g,r} \rangle, \tag{7}$$

where each anticommuting pair $X_{g,i}$ and $Z_{g,i}$ represents one gauge degree of freedom. If g and s denote the numbers of independent nonphase generators of $\mathcal{G}$ and $\mathcal{S}$, respectively, then

$$r = \frac{g - s}{2}, \qquad k = n - s - r. \tag{8}$$

The r anticommuting pairs describe the gauge degrees of freedom in a canonical basis. In a measurement implementation, one may instead choose a different generating set of $\mathcal{G}$ whose products recover the stabilizer generators. A compatible measurement schedule then allows the stabilizer syndrome to be inferred from those gauge outcomes.

The subsystem structure also leads to two notions of logical operators. Operators that commute with the full gauge group are called *bare* logical operators, whereas operators that commute with the

stabilizer group but may act nontrivially on the gauge subsystem are called *dressed* logical operators. Their equivalence classes are represented by

$$C_{\mathcal{P}_n}(\mathcal{G})/(\Phi\mathcal{S}) \qquad \text{and} \qquad C_{\mathcal{P}_n}(\mathcal{S})/\mathcal{G}, \tag{9}$$

respectively. Since a dressed logical operator may act arbitrarily on the gauge subsystem, the distance relevant to the protected logical information is the minimum weight of a nontrivial dressed logical operator:

$$d = \min_{P \in C_{\mathcal{P}_n}(\mathcal{S}) \setminus \mathcal{G}} \mathrm{wt}(P). \tag{10}$$

For a CSS subsystem code, let $G_X \in \mathbb{F}_2^{m_X \times n}$ and $G_Z \in \mathbb{F}_2^{m_Z \times n}$ contain the submitted X- and Z-type gauge measurement generators as rows, including dependent rows. Their commutation matrix is

$$\Omega = G_X G_Z^\top \tag{11}$$

over $\mathbb{F}_2$. Its nonzero entries identify anticommuting measurement pairs, and $r = \mathrm{rank}(\Omega)$. Thus $\Omega = 0$ recovers the CSS stabilizer case.

Let the rows of U_X and U_Z span the full coefficient kernels

$$U_X \Omega = 0, \qquad \Omega U_Z^\top = 0. \tag{12}$$

The products $U_X G_X$ and $U_Z G_Z$ span all X- and Z-type central supports. We choose H_X, H_Z as independent row bases of these products:

$$\mathrm{row}(H_X) = \mathrm{row}(U_X G_X), \qquad \mathrm{row}(H_Z) = \mathrm{row}(U_Z G_Z). \tag{13}$$

They satisfy $H_X H_Z^\top = 0$. Removing dependent or zero rows preserves the center without changing the submitted measurement lists.

For a CSS subsystem code, the dressed logical distance can be evaluated separately in the two Pauli sectors. In binary notation, d_X is the minimum weight of an X-type vector that commutes with every Z-type stabilizer but does not belong to the row space of G_X. Similarly, d_Z is obtained by exchanging X and Z. The overall code distance is $d = \min(d_X, d_Z)$.

This gauge-first construction provides the structural flexibility used in our discovery framework. Rather than searching directly for the stabilizers of a target code, the framework searches for low-weight gauge generators whose supports are compatible with a given hardware connectivity graph. Products of these locally realizable gauge operators can generate stabilizers with larger support, allowing the logical code structure to extend beyond the locality of any individual measured gauge check. The hardware therefore constrains the supports of G_X and G_Z, while the stabilizer structure (H_X, H_Z) and logical parameters (k, d) emerge from the selected gauge group. This separation makes the gauge generators natural design variables for adapting subsystem codes to irregular or augmented superconducting connectivity.

A.3 Superconducting quantum processors and hardware connectivity

Surface-code hardware mapping. Superconducting quantum processors typically consist of a fixed array of physical qubits connected through a hardware-defined network of native two-qubit couplings. For the purpose of quantum-code design, we represent such a processor by an interaction graph $H_0 = (V_0, E_0)$, where each vertex $v \in V_0$ corresponds to a physical qubit and each edge $(u, v) \in E_0$ denotes an available native two-qubit interaction. The degree of a vertex gives the number of physical couplers connected to the corresponding qubit, while the global structure of H_0 constrains which multi-qubit checks can be implemented using only native local interactions. Fig. 1a of the main text illustrates the connectivity of the representative 105-qubit superconducting processor considered here.

Surface codes are particularly well matched to this hardware model. A rotated planar surface code arranges data and measurement qubits on a two-dimensional local lattice. Each bulk stabilizer acts on four neighboring data qubits and can be measured using only nearest-neighbor two-qubit interactions. In a standard dedicated-ancilla layout, a distance-d rotated surface code encodes one logical

qubit using d^2 data qubits and $d^2 - 1$ measurement qubits. As illustrated in Fig. 1b of the main text, a distance-seven surface-code patch therefore requires 49 data qubits and 48 measurement qubits and can be naturally embedded into a processor at the scale considered here. Such a distance-seven surface-code memory has already been experimentally realized on a 105-qubit superconducting processor (Google, 2025).

A.4 Distance bound for the native Willow topology

We establish a distance bound for the static, direct-measurement model used in this work. The argument first reduces a local subsystem code to an ordered sequence of blocks and then uses the seven-site layers of the native Willow graph.

Model and notation. Let $H_0 = (V, E_0)$ denote the native graph with 105 sites and 182 couplers. Fix disjoint sets $D, A \subseteq V$ of data and measurement-ancilla sites, respectively, and set $n = |D|$. Each measured Pauli generator γ is assigned an ancilla $a \in A$ and satisfies

$$\operatorname{supp}(\gamma) \subseteq D \cap N_{H_0}(a), \tag{14}$$

where $N_{H_0}(a)$ is the neighborhood of a. Thus each generator is measured through one native ancilla star, with fixed data and ancilla roles.

We represent Pauli operators modulo phase by $\mathcal{P} = \mathbb{F}_2^{2n}$. For $u = (x, z)$ and $v = (x', z')$, the symplectic form is $\langle u, v\rangle = x \cdot z' + z \cdot x'$. For $U \subseteq \mathcal{P}$, write $U^\perp$ for its symplectic orthogonal complement. Let G be the span of the measured generators and let $S = G \cap G^\perp$ be the full gauge center. The bare and dressed logical spaces are $G^\perp/S$ and $S^\perp/G$, respectively, each of dimension $2k$, where k is the number of protected logical qubits. For $k \geq 1$, the dressed distance is

$$d = \min_{\ell \in S^\perp \setminus G} |\operatorname{supp}(\ell)|. \tag{15}$$

These conventions include stabilizer codes as the case $G = S$.

Theorem A.1 (Native-Willow distance bound)**.** *Every code satisfying the model above obeys*

$$k \geq 1 \quad \Longrightarrow \quad d \leq 7. \tag{16}$$

The same statement holds after deleting any subset of native sites or couplers.

To prove the theorem, we use two intermediate results. The first removes the support of a logical operator from a region by adding a gauge operator. The second uses the locality of the gauge generators to confine a nontrivial logical operator to one block.

Gauge cleaning. For $M \subseteq D$, let $\mathcal{P}_M$ be the Pauli subspace supported in M, embedded in $\mathcal{P}$ by zero extension, and let π_M denote restriction to M. Orthogonality within $\mathcal{P}_M$ is denoted by $\perp_M$.

Lemma A.2 (Gauge cleaning)**.** *If $G^\perp \cap \mathcal{P}_M \subseteq S$, then for every $\ell \in S^\perp$ there is $g \in G$ such that $\pi_M(\ell + g) = 0$.*

Proof. The restriction map satisfies

$$(\pi_M G)^{\perp_M} = G^\perp \cap \mathcal{P}_M, \tag{17}$$

because $\langle x, \pi_M g\rangle = \langle x, g\rangle$ for every $x \in \mathcal{P}_M$ and $g \in G$. By the hypothesis, the right-hand side is contained in S. Hence $\ell \in S^\perp$ implies that $\pi_M \ell$ is orthogonal to $(\pi_M G)^{\perp_M}$. The restricted symplectic form is nondegenerate, so

$$\pi_M \ell \in \left((\pi_M G)^{\perp_M}\right)^{\perp_M} = \pi_M G. \tag{18}$$

Choosing $g \in G$ with $\pi_M g = \pi_M \ell$ proves the claim. □

An ordered-block bound. The next result applies to an open sequence of blocks: generators may intersect neighboring blocks, with no periodic wrap-around coupling.

Lemma A.3 (Open-strip bound)**.** *Let $D = R_0 \sqcup \cdots \sqcup R_{L-1}$, where $L \geq 1$ and $|R_i| \leq b$ for every i. Suppose G has a generating set Γ such that every $\gamma \in \Gamma$ is supported in one block or in two consecutive blocks. If $k \geq 1$, then $d \leq b$.*

Proof. Write $M_t = \bigcup_{i=0}^{t} R_i$. Since $M_{L-1} = D$ and $k \geq 1$, there is a smallest index t such that M_t supports an operator $\ell \in S^\perp \setminus G$. If $t = 0$, its weight is at most $|R_0| \leq b$, which proves the result.

Consider $t > 0$. Minimality gives $S^\perp \cap \mathcal{P}_{M_{t-1}} \subseteq G$. Using $G^\perp \subseteq S^\perp$ and $G \cap G^\perp = S$, we obtain

$$G^\perp \cap \mathcal{P}_{M_{t-1}} \subseteq S. \tag{19}$$

Lemma A.2 therefore provides $g \in G$ with $\pi_{M_{t-1}} g = \pi_{M_{t-1}} \ell$.

Expand $g = \sum_{\gamma \in \Gamma} c_\gamma \gamma$, with $c_\gamma \in \mathbb{F}_2$, and retain only terms that meet M_{t-1}:

$$g' = \sum_{\gamma \in \Gamma:\ \mathrm{supp}(\gamma) \cap M_{t-1} \neq \varnothing} c_\gamma \gamma. \tag{20}$$

The omitted terms vanish on M_{t-1}, so g' performs the same cleaning there. Every retained term is supported in M_t: it meets the prefix M_{t-1} and spans at most two consecutive blocks. Consequently,

$$\mathrm{supp}(\ell + g') \subseteq R_t. \tag{21}$$

Finally, $g' \in G \subseteq S^\perp$ implies $\ell + g' \in S^\perp$, while $\ell \notin G$ implies $\ell + g' \notin G$. Thus $\ell + g'$ is a nontrivial dressed logical operator of weight at most $|R_t| \leq b$. □

Proof of Theorem A.1. Let (x_v, y_v) be the integer coordinates of site v in the native graph and define $t(v) = x_v + y_v$. Direct inspection of its coordinate and edge lists gives

$$\begin{aligned} t(V) &= \{6, 7, \ldots, 20\}, \\ |\{v \in V : t(v) = j\}| &= 7 \qquad (6 \leq j \leq 20), \\ |t(u) - t(v)| &= 1 \qquad (\{u, v\} \in E_0). \end{aligned} \tag{22}$$

We now use these properties to verify the hypotheses of Lemma A.3.

Parity decomposition. All neighbors of an ancilla a lie in layers $t(a) - 1$ or $t(a) + 1$. Thus every generator in Eq. 14 acts on data sites of a single parity. Define $D_p = \{v \in D : t(v) \equiv p \pmod 2\}$ for $p \in \{0, 1\}$, and let G_p be the span of the generators supported in D_p. Since D_0 and D_1 are disjoint,

$$G = G_0 \oplus G_1, \qquad S = S_0 \oplus S_1, \qquad k = k_0 + k_1, \tag{23}$$

where $S_p = G_p \cap G_p^\perp$ and k_p are computed in the Pauli space on D_p. Hence $k \geq 1$ implies $k_p \geq 1$ for at least one sector.

Seven-site blocks. Within that sector, order the layers of parity p by increasing t, including any empty layers, and intersect them with D_p. Eq. 22 bounds each resulting block by seven data sites. A generator anchored at a meets only the layers $t(a) - 1$ and $t(a) + 1$, which are consecutive in this order. Lemma A.3, with $b = 7$, gives a nontrivial dressed logical operator of weight at most seven on D_p. Its zero extension is nontrivial in $S^\perp / G$, proving Eq. 16.

Deleting sites or couplers preserves the layer-size bound and the single-star support condition, so the argument also applies to every deletion subgraph. □

The bound identifies a geometric limitation of native connectivity even when data placement and gauge measurements are optimized jointly. Within the same static, direct-measurement model, obtaining $d \geq 8$ requires connections beyond E_0. This motivates optimizing connectivity while retaining the same physical-site budget.

Locality and hardware limitations. The strong compatibility between surface codes and two-dimensional local hardware follows from general constraints on geometrically local quantum codes. For codes defined by local commuting constraints in two dimensions, the code parameters obey the tradeoff $kd^2 = O(n)$ (Bravyi et al., 2010). Planar surface codes with open rough and smooth boundaries achieve $kd^2 = \Theta(n)$ and therefore saturate this bound up to constant factors. Within this class of codes, surface codes thus make asymptotically optimal use of physical qubits when the hardware provides the regular two-dimensional connectivity assumed by the code. However, this locality bound applies to local commuting stabilizer constraints and does not directly restrict subsystem codes whose local gauge generators may produce stabilizers with larger or nonlocal support.

At the circuit level, the local geometry of the surface code also enables regular and efficient syndrome extraction. A fault on a measurement qubit can propagate through successive entangling gates and produce a correlated data-qubit hook error. However, the entangling gates can be ordered so that these hook errors do not align with the corresponding logical operators. The rotated surface code therefore admits distance-preserving syndrome-extraction schedules that prevent hook errors from reducing the effective circuit-level distance (Tomita & Svore, 2014).

Together, these properties make the surface code a particularly strong solution for an ideal two-dimensional superconducting lattice. Our goal is not to improve upon this regime. Instead, we consider hardware graphs that depart from the ideal lattice, either because some physical qubits or couplers are unavailable or because a limited number of additional nonlocal couplers can be introduced. In these settings, the separation between low-weight gauge measurements and the stabilizers they generate gives subsystem codes additional structural flexibility. The specific defect and connectivity-augmentation models considered in this work are introduced in the Numerical Results section.

B Extended related works

B.1 Subsystem codes

Existing constructions. Subsystem quantum error correction was established through the operator-QEC framework, which protects logical information in a subsystem while introducing additional gauge degrees of freedom (Kribs et al., 2005; Poulin, 2005). One of the best-known early constructions is the Bacon–Shor code, which uses the subsystem structure to simplify error correction (Bacon, 2006; Aliferis & Cross, 2007). Subsequent work further exploited gauge freedom to reduce measurement weight and improve locality. Topological subsystem codes enabled syndrome extraction using low-weight gauge measurements, while generalized Bacon–Shor codes and subsystem surface codes introduced spatially local gauge operators and three-qubit check measurements (Bombín, 2010; Bravyi, 2011; Bravyi et al., 2012).

More hardware-oriented constructions have explicitly targeted limited qubit connectivity. A prominent example is the heavy-hexagon code, which combines features of surface and Bacon–Shor codes on a low-degree graph motivated by the connectivity constraints of superconducting processors (Chamberland et al., 2020b). These works demonstrate the flexibility of subsystem-code design. However, most rely on carefully constructed code families with prescribed algebraic or lattice structures, motivating a more systematic search for subsystem codes under hardware-specific constraints.

Automated code search. Computational search provides an alternative to manually constructing subsystem codes. An early and particularly relevant approach developed an algorithm for computing the optimal subsystem code supported by a given set of Pauli measurement operators and used it to systematically investigate two-body nearest-neighbor measurements on lattices derived from convex uniform tilings of the plane (Crosswhite & Bacon, 2011). This provided an early demonstration that subsystem-code design can be approached computationally under explicit locality and connectivity constraints.

More recent work has expanded automated QEC design through learning and optimization. Reinforcement learning (RL) has been used to adapt surface-code memories to different error models and connectivity settings (Nautrup et al., 2019), while variational methods have jointly searched for quantum codes and hardware-efficient encoding circuits under specified noise and connectivity constraints (Cao et al., 2022). RL has also been applied to structured code spaces based on tensor networks and Quantum Lego, optimizing objectives such as code distance and logical performance (Mauron et al., 2024; Su et al., 2025). Evolutionary and reinforcement-learning methods have further been used to search for stabilizer codes tailored to particular error models and to discover low-weight code constructions (Webster & Browne, 2025; He & Liu, 2025).

Automated discovery has increasingly incorporated hardware and implementation constraints. RL has been used to jointly discover QEC codes and encoding circuits for specified gate sets, qubit connectivity, and error models (Olle et al., 2024), and to tailor BB codes to a fixed decoder architecture (Sella et al., 2026). More recently, large language models have been introduced into

automated QEC discovery. LLM-guided evolutionary search has explored bivariate-bicycle and perturbed bivariate-bicycle code families with independent candidate validation (Cruz-Benito et al., 2026). OmniQEC further formulates QEC design as an AI-scientist-driven iterative process that combines code generation, screening, syndrome-extraction synthesis, and decoder-based circuit evaluation across multiple qLDPC construction families (Yan et al., 2026). In comparison, direct automated search over subsystem gauge structures, particularly under prescribed and irregular hardware connectivity, remains comparatively underexplored.

B.2 Automatic search for QEC codes

RL-based methods. RL has emerged as a flexible approach for navigating the large combinatorial search spaces involved in QEC design. Early work used RL to adapt surface-code quantum memories to different error models and arbitrary qubit connectivity, demonstrating that code layouts can be optimized automatically for a target physical setting (Nautrup et al., 2019). RL was later extended to structured stabilizer-code spaces, including tensor-network constructions and Quantum Lego, where the search objective can target properties such as code distance or logical error probability (Mauron et al., 2024; Su et al., 2025). These studies demonstrate that RL can efficiently explore structured code spaces beyond direct enumeration.

More recent RL-based approaches have incorporated increasingly explicit implementation constraints. RL has been used to jointly discover QEC codes and their encoding circuits for specified gate sets, qubit connectivity, and error models (Olle et al., 2024). Other work has targeted low-weight stabilizer codes (He & Liu, 2025), while decoder-aware RL has been used to tailor BB codes to a fixed decoder architecture (Sella et al., 2026). Together, these works show that RL-based code discovery can incorporate code properties, noise models, connectivity constraints, and downstream decoding performance into the search objective.

LLM- and agent-based methods. Large language models have recently been introduced as generative search agents for QEC code discovery. LLM-guided evolutionary search has been used to mutate executable programs that generate bivariate-bicycle and perturbed bivariate-bicycle code candidates, with deterministic tools used to evaluate and verify the resulting codes (Cruz-Benito et al., 2026). Another approach, structured concept evolution, combines an LLM with a structured algebraic mutation grammar to explore lifted-product CSS qLDPC code families (Liu & Marquardt, 2026). LLM-assisted heuristic search has also been used to identify lifted-product code instances for fault-tolerant quantum architectures (Cain et al., 2026). These studies show that LLMs can guide the structured exploration of large code spaces while external tools evaluate candidate validity and performance.

More recent work has extended this idea toward longer and more implementation-aware discovery loops. OmniQEC formulates QEC design as an AI-scientist-driven iterative process that combines code generation, code-level screening, syndrome-extraction synthesis, and decoder-based circuit evaluation across multiple qLDPC construction families (Yan et al., 2026). Evaluation results are fed back into subsequent iterations, allowing the search to refine candidates using both code-level and implementation-level criteria. Agent-based approaches have also been used to search for practical finite-length qLDPC codes through specialist proposal and review, executable-program evolution, and deterministic evaluation (Qian & Li, 2026). These frameworks provide another route toward more autonomous and iterative code discovery.

C Candidate validity and the full gauge center

C.1 Measurement Realizations and Resource Accounting

A candidate declares at least one data site and one ancilla site. The sets D, A, U partition the available physical qubits V; an omitted U is recovered as the complement of $D \cup A$. Site labels and coordinates come from the hardware input, and roles remain fixed throughout a realization.

For each Pauli type $\beta \in \{X, Z\}$, index the m_β submitted measurements by $j = (\beta, i)$, where $1 \le i \le m_\beta$. Let g_j and $\alpha(j) \in A$ denote the measured generator and its ancilla, respectively. Define the measurement lists

$$\mathcal{M}_\beta = \big\{(\beta, \alpha((\beta, i)), \mathrm{supp}(g_{(\beta,i)})) : i = 1, \dots, m_\beta\big\}, \qquad \beta \in \{X, Z\}. \tag{24}$$

Each triple (β, a, W) specifies a Pauli type, an ancilla anchor, and a nonempty data support. It measures $\prod_{q\in W} \beta_q$ and requires a direct ancilla–data edge to every $q \in W$. An ancilla may serve several measurements.

The used ancillas and occupied physical qubits of a direct-measurement realization are

$$\begin{aligned} A^\circ(C) &= \{\alpha(j) : j = (\beta, i), \quad \beta \in \{X, Z\},\ 1 \le i \le m_\beta\}, \\ Q(C) &= |D| + |A^\circ(C)|. \end{aligned} \tag{25}$$

We separately record the nominal budget $|V_0|$, available physical qubits $|V|$, data-qubit count $n = |D|$, declared ancillas $|A|$, and used ancillas $|A^\circ(C)|$. For routed circuits, Q additionally includes every distinct routing site touched by the circuit. Couplers are counted as distinct edges, while measurement counts and support weights use the submitted lists.

Each measurement triple (β, a, W) in $\mathcal{M}_\beta$ must be unique. Identical supports at different anchors and distinct but linearly dependent measurements are permitted. The physical lists describe the measurement realization, whereas independent gauge bases describe its algebraic structure. Validation proceeds through format, labels, roles, measurement types and supports, followed by weight limits, duplicate checks, measurement-count limits, and connection constraints. Gauge algebra and distance are evaluated after these checks succeed. Each execution record identifies its outcome and completed validation stages. Algebraic evaluation includes every declared data site and uses the full measurement lists returned by the program.

C.2 Binary Representation and Code Parameters

We use phase-free binary Pauli supports over $\mathbb{F}_2$, with data columns ordered by site label. The submitted matrices are

$$G_X \in \mathbb{F}_2^{m_X \times n}, \qquad G_Z \in \mathbb{F}_2^{m_Z \times n}. \tag{26}$$

Let $q_\beta = \operatorname{rank}(G_\beta)$ for $\beta \in \{X, Z\}$. Row reduction gives independent bases $\bar{G}_\beta \in \mathbb{F}_2^{q_\beta \times n}$ with

$$\mathcal{G}_\beta = \operatorname{im}(G_\beta^\top) = \operatorname{im}(\bar{G}_\beta^\top). \tag{27}$$

Dependent measurements still contribute to the resources of their physical realization.

The basis commutation matrix $\bar{\Omega} = \bar{G}_X \bar{G}_Z^\top$ has the same rank as the submitted-matrix commutation matrix Ω. The restricted symplectic form has rank $2r$, where $r = \operatorname{rank}(\bar{\Omega})$. If the full center has dimension s, then $q_X + q_Z = s + 2r$ and $n = k + r + s$. Hence

$$r = \operatorname{rank}(\Omega) = \operatorname{rank}(\bar{\Omega}), \qquad k = n - q_X - q_Z + r. \tag{28}$$

C.3 Computing the Full Center

For $u \in \mathbb{F}_2^{q_X}$, the support $\bar{G}_X^\top u$ commutes with every Z gauge operator exactly when $\bar{\Omega}^\top u = 0$. Similarly,

$$\begin{aligned} \mathcal{S}_X &= \{\bar{G}_X^\top u : u \in \mathbb{F}_2^{q_X},\ \bar{\Omega}^\top u = 0\}, \\ \mathcal{S}_Z &= \{\bar{G}_Z^\top v : v \in \mathbb{F}_2^{q_Z},\ \bar{\Omega} v = 0\}. \end{aligned} \tag{29}$$

Mapping bases of these full coefficient kernels into data-support space gives bases for the full center. Let $s_\beta = \dim \mathcal{S}_\beta$ for $\beta \in \{X, Z\}$. Their transposes form the rows of $H_X \in \mathbb{F}_2^{s_X \times n}$ and $H_Z \in \mathbb{F}_2^{s_Z \times n}$, where $\operatorname{im}(H_\beta^\top) = \mathcal{S}_\beta$. These are the same center spaces defined from the submitted matrices in the preliminaries.

The verifier checks

$$s_X = q_X - r, \qquad s_Z = q_Z - r, \qquad s = s_X + s_Z, \qquad n = k + r + s, \tag{30}$$

together with $k \ge 0$, $H_X G_Z^\top = 0$, and $G_X H_Z^\top = 0$. Records retain the submitted matrices, gauge bases, center bases, commutation matrices, and data-column map.

D Distance Evaluation and Evidence

D.1 Reduction to Two Pure Directions

A nontrivial dressed logical operator commutes with the full center and lies outside the gauge group. In the CSS representation, its support (x, z) satisfies $H_Z x = 0$, $H_X z = 0$, and $(x, z) \notin \mathcal{G}_X \oplus \mathcal{G}_Z$. At least one component lies outside its gauge support space and is therefore a nontrivial pure-direction logical operator. Its weight is no larger than the mixed Pauli support. Conversely, a pure-direction logical is a dressed logical. Thus, for $k > 0$,

$$\begin{aligned} d_X &= \min_{x \in \ker H_Z \setminus \mathcal{G}_X} \mathrm{wt}(x), \\ d_Z &= \min_{z \in \ker H_X \setminus \mathcal{G}_Z} \mathrm{wt}(z), \qquad d = \min(d_X, d_Z). \end{aligned} \tag{31}$$

Candidates with $k = 0$ receive the status `no_logicals`.

D.2 A Linear Test for Logical Classes

Consider the X direction. We have $\mathcal{G}_X \subseteq \ker H_Z$ and

$$\dim(\ker H_Z) - q_X = n - (q_Z - r) - q_X = k. \tag{32}$$

Choose $L_X \in \mathbb{F}_2^{k \times n}$ so that the columns of $\bar{G}_X^\top$ and $L_X^\top$ together form a basis of $\ker H_Z$. Let T_X be the resulting full-row-rank matrix and R_X a right inverse. Writing I_j for the $j \times j$ identity, define the quotient-coordinate map Π_X by

$$\begin{aligned} &T_X = \begin{bmatrix} \bar{G}_X \\ L_X \end{bmatrix} \in \mathbb{F}_2^{(q_X+k)\times n}, \\ &R_X \in \mathbb{F}_2^{n \times (q_X+k)}, \qquad T_X R_X = I_{q_X+k}, \\ &\Pi_X = [0_{k \times q_X} \quad I_k]\, R_X^\top \in \mathbb{F}_2^{k \times n}. \end{aligned} \tag{33}$$

Every $x \in \ker H_Z$ has unique coordinates $c \in \mathbb{F}_2^{q_X+k}$ such that $x = T_X^\top c$. Since $R_X^\top x = c$, the map Π_X extracts the last k coordinates, corresponding to the added basis vectors. It follows that

$$x \in \ker H_Z \quad \Longrightarrow \quad \big(\Pi_X x = 0 \iff x \in \mathcal{G}_X\big). \tag{34}$$

The verifier checks that Π_X vanishes on the gauge space and has full-rank action on the added quotient basis. A logical-support test then reduces to $H_Z x = 0$ and $\Pi_X x \neq 0$. Exchanging X and Z gives Π_Z.

D.3 Logical Witnesses and Weight Enumeration

For each direction, we first examine a basis of the relevant kernel ($\ker H_Z$ for X and $\ker H_X$ for Z) and controlled random combinations of its vectors. Every support identified as a nontrivial dressed logical operator supplies a constructive upper bound on distance.

We then enumerate data supports in increasing weight $w = 1, 2, \ldots$. A completed scan that excludes nontrivial logical operators at every weight from 1 through w raises the lower bound to $w + 1$. If all smaller weights have been excluded and a logical witness is found at weight w, the directional distance equals w. An interrupted weight layer retains the lower bound from the last completed layer. The weight-one scan is mandatory.

Let τ be the configured enumeration threshold and $\overline{d}_\alpha$ an available upper bound for direction $\alpha \in \{X, Z\}$. Enumeration normally targets weights through $\tau - 1$. For $n \leq 24$ with an available upper bound, the target can extend to $\max(\tau - 1, \overline{d}_\alpha)$. Enumeration above weight one is subject to the specified support-count and time budgets; τ controls its computational range.

D.4 Mixed-Integer Linear Programming

When enumeration leaves the distance unresolved and $n \leq 105$, evaluation continues through SciPy's MILP interface. For the X direction, set $M = H_Z$ and $\Pi = \Pi_X$; for the Z direction,

set $M = H_X$ and $\Pi = \Pi_Z$. Let m be the number of rows of M. Lift the binary matrices to integer matrices with entries in $\{0, 1\}$, so that $M \in \{0, 1\}^{m \times n}$ and $\Pi \in \{0, 1\}^{k \times n}$. Introduce a support $x \in \{0, 1\}^n$, logical coordinates $y \in \{0, 1\}^k$, and integer variables $a \in \mathbb{Z}^m_{\geq 0}$, $b \in \mathbb{Z}^k_{\geq 0}$. The distance problem becomes

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} x_i \\ \text{s.t.} \quad & Mx - 2a = 0, \\ & \Pi x - y - 2b = 0, \\ & 1 \leq \sum_{j=1}^{k} y_j \leq k. \end{aligned} \tag{35}$$

The first two constraints encode binary parity relations over the integers, and the final constraint selects a nonzero logical class.

Each returned support is converted to a binary vector and rechecked for $Mx = 0$ and $\Pi x \neq 0$ over $\mathbb{F}_2$. A verified witness can tighten the upper bound. An MILP result is recorded as the exact directional distance when the solver reports optimality and the revalidated witness weight agrees with the objective value to within 10^{-6}. For other solver statuses, verified feasible solutions supply upper bounds. Floating-point dual bounds and gaps are retained as solver diagnostics. Exact distance can also follow from completed enumeration or matching bounds.

The solver reports MILP optimality, while witness validity is checked independently over $\mathbb{F}_2$. When evaluation returns normally, previously established evidence is retained through solver errors or timeouts. A distance worker that fails to return receives an unknown evaluation status, and recovery uses persisted records.

D.5 Combining Directional Evidence

Let $\underline{d}_\alpha, \overline{d}_\alpha$ denote the available bounds for direction $\alpha \in \{X, Z\}$. When both directional bounds are available,

$$\underline{d} = \min(\underline{d}_X, \underline{d}_Z), \qquad \overline{d} = \min(\overline{d}_X, \overline{d}_Z). \tag{36}$$

A total lower bound requires lower bounds in both directions. An upper bound in either direction also bounds the total distance; when just one is available, that bound is used. Unresolved endpoints retain an unknown status.

Matching bounds establish the exact total distance, and $\underline{d} \geq d_\star$ certifies the distance target. For example, one direction with exact distance 5 and the other with a lower bound of at least 5 jointly establish $d = 5$. Consistency requires $\underline{d} \leq \overline{d}$; a conflicting interval triggers an evidence-conflict status and suspends its use in ranking.

D.6 Bounded Distance Upgrades

When enabled, distance upgrades allocate additional classical computation to candidates whose evidence can be strengthened. For a task with thresholds $k_\star, d_\star$, the pool consists of current and historical evaluated candidates satisfying the general parent criteria and $k \geq k_\star$. Priority follows the distance evidence, and each labeled gauge group contributes one solve target.

The implementation upgrades at most two gauge groups per round and each group at most once per run. An upgrade seeks to certify a distance lower bound above the current one and consistent with any available upper bound. For distance-maximization tasks, this value is at least the current lower bound plus one, allowing evidence to improve beyond the initial task threshold. Each upgrade receives a longer, prespecified solver time budget.

Budget reservations are persisted before invocation, with failures and interruptions counted against the quota. Evidence is merged for candidates with identical data labels and full gauge spaces, taking the maximum valid lower bound and minimum valid upper bound in each direction. Updates are stored as separate records associated with their rounds, preserving both the original execution and the time at which stronger evidence became available.

D.7 A Necessary Condition from Center Rank

For $k > 0$, the center ranks imply

$$d_X \le s_Z + 1, \qquad d_Z \le s_X + 1, \qquad d \le \min(s_X, s_Z) + 1. \tag{37}$$

To see this, put H_Z, whose rank is s_Z, in systematic form using row operations and a column permutation. Its kernel admits a basis whose vectors have weight at most $s_Z + 1$. If every basis vector belonged to $\mathcal{G}_X$, then the entire kernel would lie in $\mathcal{G}_X$, contradicting the positive quotient dimension in Eq. 32. At least one basis vector therefore represents a nontrivial pure X logical operator. Exchanging the two directions gives the remaining inequalities.

A target distance $d_\star$ consequently requires $s_X, s_Z \ge d_\star - 1$. This necessary condition supplies a structural diagnostic for feedback.

E Program Search Details

E.1 Round Structure and Construction Interface

Let $\mathcal{A}_{t-1}$ contain the fully committed execution records and evidence available at the start of round t. A fresh run starts with an empty archive and distance cache; a resumed run reconstructs them from saved state. Let P be the configured upper limit on accepted programs per round, $P_t \le P$ the number accepted in round t, and S the number of seeds evaluated per program. Each round proceeds as follows:

1. Select parents from $\mathcal{A}_{t-1}$ and assemble the previous round's program summaries and recent failure examples, together with bounded historical observations when memory is enabled.
2. Request construction programs, parse the response, check parent identifiers, and remove duplicate source programs. Accept $P_t \le P$ programs. Configured completion requests can supplement an underfilled batch, after which a nonempty batch is executed at its actual size. On transport failure or an empty accepted batch, save state and end the run.
3. Execute each accepted program with S seeds. Check format, roles, measurements, hardware constraints, and gauge algebra, retaining the execution outcome or the candidate for distance evaluation.
4. Reuse or compute distance evidence by labeled gauge-group identity. Apply enabled upgrades within their separate classical budget.
5. Save all execution results, program summaries, and evidence updates. Commit the round to form $\mathcal{A}_t$, then prepare the next references.

All programs in a round share the reference pool. Records distinguish raw model output, accepted programs, and actual executions, and track completion requests, transport retries, and new-round proposals separately.

The model returns a JSON object containing source code and parent identifiers. Each program implements `construct(rng, chip)`, which returns a candidate. The controlled random generator `rng` supplies construction randomness, while `chip` exposes hardware physical qubits, connection information, and task constraints. Programs may build candidates independently or develop the supplied references, including constructions from known code families. The classical stages evaluate the submitted constructions.

Before execution, the system checks the function signature, permitted syntax, call whitelist, source size, and syntax-tree size. Programs operate on the supplied inputs through permitted local operations in separate processes with output-size and wall-clock limits. Construction exceptions, timeouts, hardware rejections, and distance-worker failures retain distinct statuses.

Round indices start at $t = 1$, program indices are $j = 0, \dots, P_t - 1$, and seed indices are $s = 0, \dots, S - 1$. Each execution initializes a Python random generator with

$$\sigma_{t,j,s} = \sigma_0 + 100000t + 1000j + s, \qquad \sigma_0 = 20260908. \tag{38}$$

The recorded mapping makes construction inputs traceable by round, program, and seed.

E.2 Distance Caching and Realization Identity

The distance-cache identity is

$$\kappa(C) = \big(D, \mathcal{G}_X, \mathcal{G}_Z\big), \tag{39}$$

where D is ordered by site label and both spaces are represented by canonical row bases. Each seed execution constructs its candidate and passes hardware and gauge-algebra checks before consulting the distance cache. Candidates with different generator orderings, redundant rows, or ancilla assignments can share this identity. Their measurement lists, connections, and resource records remain separate.

Cache identity uses the full spaces in a fixed data labeling and X/Z orientation. The resulting distinct-group count therefore measures labeled gauge groups. Hardware and resource evaluation apply to each measurement realization as submitted by its construction program.

E.3 Feedback, Reference Selection, and Memory

Program summaries. Summaries cover all programs from the previous round. They record execution counts, status distributions, and the numbers of distinct data sets, role assignments, supports, anchored measurements, required edges, added edges, and labeled gauge groups. These statistics describe diversity in the actual outputs.

Each round includes at most four recent failure examples, prioritizing coverage of failure mechanisms and then of programs and realizations. When enumeration or MILP has produced stored logical witnesses, feedback can select and revalidate at most one per direction. For a parent and child with identical data sets, an old witness is rechecked against the child's center and gauge space. The result describes whether that specific logical action persists in the child; the child's distance follows from its full distance evaluation.

Final ranking and reference selection. Task winners follow the query-conditioned ranking described in Section 3, while reference selection maintains candidates for further exploration. All parents pass hardware and algebra checks and satisfy $k \geq 1$ and $\underline{d} \geq 2$.

In protection-first lexicographic mode, the pool retains the task winner and distance winner, supplemented by recent structurally distinct exploratory references. In target-constrained mode, it retains the target-feasible objective winner, a distance reference satisfying the required k, a capacity reference satisfying the required distance, and other protected branches showing distance or capacity progress. This separates final task attainment from progress toward its protection thresholds.

The pool contains at most 12 parents, including at most 4 exploratory references. For capacity or distance objectives, exploratory references require $\underline{d} \geq \max(2, d_\star - 1)$. Other modes retain references within a distance-lower-bound gap of one from the current distance winner. Distance optimization prioritizes routes satisfying the required logical capacity. The actual pool size follows the available eligible records.

For a fixed gauge group and protection evidence, the pool can use an evaluated realization with a better primary objective. When distinct structures tie on that objective, one reference slot can rotate with the round index while preserving the tie in final ranking. Global X/Z exchange may be folded during context selection, with verification and distance caching retaining the identity defined above. When the pool has spare capacity after selecting distinct gauge groups, it can include additional anchored realizations of the same group.

Bounded memory. When enabled, memory is reconstructed from committed records as recent program summaries, progress records, historical observations, and ancestor links. Each observation carries supporting record identifiers and an explicit scope. Examples include a program repeatedly generating the same gauge group, structural invariance after referencing a parent, and center ranks that fall short of the necessary distance condition. Memory is assembled directly from the archive.

The memory budget retains summaries from the last three rounds, at most six progress records, eight historical observations, and three ancestor links per parent. Its serialized size is capped at $20{,}000$ JSON characters. Entries are included or omitted in full, with omission counts recorded. If assembly fails, memory is marked unavailable and the system uses ordinary feedback. New distance evidence can enter the current context while retaining its original availability timestamp.

E.4 Construction and Degree-Conflict Diagnostics

Optional diagnostics provide two kinds of information. A program may return a construction report containing its stopping stage and reason, counts of examined physical qubits or anchors, backtracking counts, and a local conflict. Accepted fields are restricted to specified labels, integers, and valid site identifiers, and are marked as program-reported. Separately, the verifier computes installed-degree conflicts, including the site, native and installed degrees, installed and added neighbors, and associated measurement identifiers. Up to eight conflicting physical qubits are displayed, together with an omission count.

Diagnostics accompany execution records through checkpoints, failure examples, program summaries, and memory. If a later execution in a group has a diagnostic missing from the current representative, the full representative record is replaced so that its identifier, error, and report remain aligned. Group execution counts are retained separately from the representative's local counts. Memory holds at most two diagnostic examples per failure class, subject to the total serialization budget.

Program reports describe the information returned by that execution; verifier diagnostics describe the validation stages completed for its candidate. Local conflicts and exhausted construction budgets retain their execution-specific scope. These diagnostics enrich the context supplied to subsequent proposals, while evaluation and selection follow the hardware rules, algebraic criteria, distance definition, and task objective specified above.

E.5 Commit, Recovery, and Run Records

Complete model responses and per-execution checkpoints support recovery of unfinished classical evaluations. A saved response is reused when resuming its evaluations. Round commitment is recorded separately from intermediate checkpoints, and request retries are tracked separately from local evaluation recovery.

Run metadata retain the model identifier, invocation settings, actual requests, and recovery events. These records establish the provenance of each generation batch alongside the candidate-level hardware checks and distance evidence.

F Implementation details

F.1 Calibration-informed defect instances

Table 4: Calibration values of the components explicitly removed in the defect instances. Coupler values are parallel CZ XEB Pauli errors per cycle (%); site values are T_1 (μs). A dot indicates membership.

Component	Value	2C	4C	9C	2Q+2C
$(37, 50)$	1.211	•	•	•	•
$(25, 26)$	1.157	•	•	•	•
$(80, 89)$	1.113	–	•	•	–
$(79, 80)$	1.062	–	•	•	–
$(5, 6)$	1.039	–	–	•	–
$(51, 64)$	1.038	–	–	•	–
$(38, 51)$	0.951	–	–	•	–
$(74, 83)$	0.856	–	–	•	–
$(37, 38)$	0.828	–	–	•	–
Site 77	27.39	–	–	–	•
Site 48	36.52	–	–	–	•

Fig. 5 shows the frozen public Willow calibration metrics used to construct the defect instances (Google Quantum AI, 2026b;a). Couplers are ranked by decreasing stored parallel CZ XEB Pauli error per cycle; physical qubits are ranked by increasing idle T_1. Components are removed in that order subject to retaining one connected component and introducing no articulation vertex beyond those of the intact graph. The 2C, 4C, and 9C edge sets are nested. For 2Q+2C, the two

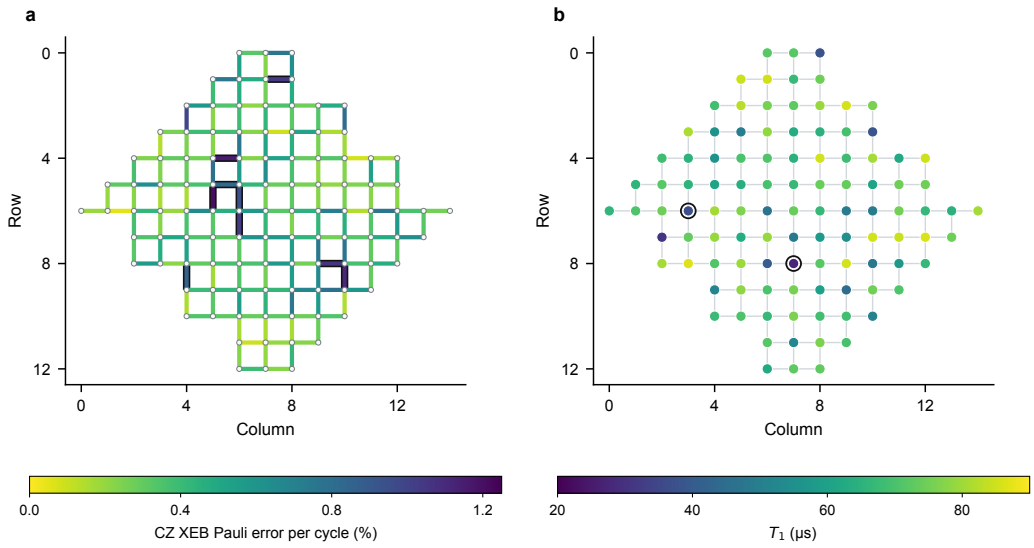

Figure 5: **Frozen calibration data on the 105-site, 182-coupler Willow graph.** **a**: parallel CZ XEB Pauli error per cycle, with the nine couplers removed in 9C outlined in black. **b**: idle T_1, with the two physical qubits removed in 2Q+2C outlined. Axes use the original Cirq grid coordinates. Both color scales are linear.

physical qubits are selected first; their eight incident edges are removed, followed by two additional couplers. Tab. 4 lists the selected components and calibration values. These masks are deterministic, calibration-informed stress tests, rather than observed simultaneous hardware failures.

F.2 Experimental configurations and continuation protocol

The connectivity tasks use the same pool of 105 physical qubits. The $\Delta \leq 5$ Willow-augmentation task maximizes d subject to $k \geq 2$. The $\Delta \leq 6$ Willow-augmentation task maximizes k among candidates. Both retain all 182 native couplers and enforce their installed-degree caps. The free-connectivity task minimizes Δ subject to $k \geq 1$ and $d \geq 9$, without requiring native-edge retention. Fig. 4a–c also includes diverse intermediate discoveries; their displayed distances need not meet the final protection target.

Tab. 5 summarizes the configurations used for the final seven-setting comparison. Each campaign used base seed 20260908, with 20 constructor seeds evaluated per program. Program counts in the table are requested batch sizes.

Table 5: Experimental configurations. Counts are per setting; programs are requested per round and seeds are evaluated per selected program.

Model	Settings	Effort	Rounds	Programs	Seeds
GPT-5.6-sol	All seven	High	60	24	20
DeepSeek-V4-Pro	All seven	Max	60	24	20
Claude Opus 5	All seven	Medium	60	24	20

For the Claude connectivity runs, adaptive thinking used an output limit of 128,000 tokens per request segment, including thinking and text. At a token-limit stop, the native assistant content blocks, including thinking blocks and their signatures, were retained in the same-round message history, followed by a continuation request. The limit was eight segments including the initial request; completed rounds used one to four. Complete programs were extracted from the resulting text and accumulated without duplicate source code.

Table 6: Circuit fault-distance certificates for the routed surface-code baselines with static distance $d = 7$. The reported distance is the minimum over the two memory bases; both combined values are exact.

Defects	X memory	Z memory	d_c
2C	6	$[6, 7]$	6
4C	6	6	6

Usage accounting. Token totals include the recorded generation and continuation calls. GPT and Claude costs use the corresponding fixed API prices. DeepSeek's recorded total for the seven tasks is CNY 1059.19 (USD 156.94 at CNY 6.7489 per USD); its per-task costs are allocated in proportion to reference-price estimates. Thus the plot summarizes the recorded search resources, with model-specific generation budgets reported in Tab. 5.

Baseline embedding checks. For the fixed TB and BB measurement constructions in Tab. 3, we tested unrestricted injective assignments of their data qubits and dedicated check ancillas to the 105-qubit Willow graph, retaining all native couplers. Two CP-SAT formulations enforced the degree bound on the union of native and required measurement edges. Each baseline was tested with two solver workers for 300 seconds in the first formulation and 420 seconds in the second. Both formulations returned `UNKNOWN` for TB at $\Delta \leq 5$ and BB at $\Delta \leq 6$. Thus, no feasible embedding was found within these budgets; infeasibility was not established. Explicit layouts were independently verified after relaxing the caps to 6 and 7, respectively. The check fixes the chosen original independent checks and dedicated ancillas; it does not exhaust alternative measurement bases, ancilla reuse, or routed realizations.

F.3 Scope of the surface-code baselines

Our native and routed baselines evaluate unchanged rotated surface-code templates under direct embedding and round-trip SWAP routing. Local check modifications, gauge-based superchecks, and ancilla repurposing offer additional, SWAP-free adaptation strategies (Auger et al., 2017; Leroux et al., 2025). Comparing these protocols requires constructing their measurements and schedules on the same fixed masks and evaluating the resulting distances and resources.

The selected physical qubits and couplers are unavailable throughout our experiments. Noise-aware decoding of usable but unreliable components (Drouet et al., 2026) addresses a related, distinct setting: decoder reweighting alone does not restore a missing interaction. Such decoding can complement structural adaptation once a realizable measurement circuit is specified. Our reported advantages concern the native and routed implementations evaluated here; specialized adaptation protocols remain relevant comparators.

F.4 Circuit fault-distance certificates

For each implementation, we construct separate X- and Z-basis memory circuits with $T = d$ complete measurement cycles. A fault is one nontrivial Pauli branch after a Hadamard or CZ gate, a preparation flip, or a readout flip at one elementary circuit location. The circuit fault distance is the minimum number of such faults producing no detector events and a nontrivial protected logical flip. The reported value is the smaller distance of the two memory circuits.

Lower bounds use linear projections of the detector constraints that retain every fault column and all logical labels. Removing detector constraints enlarges the allowed fault sets, so a lower bound for the projected problem remains a lower bound for the original circuit. For graphlike projections, an all-root signed-double-cover shortest-path certificate determines this lower bound. Projections containing hyperedges are treated with the full parity and fault-weight constraints. UNSAT results are checked by RUP proof replay and by verifying the detector-model-to-CNF encoding. Upper bounds use explicit fault sets replayed in the original circuit. Coinciding bounds certify an exact distance.

For the two routed baselines in Tab. 6, six-fault witnesses establish the X-memory upper bounds, while the Z-memory certificates exclude all fault sets of weight at most five. Thus both combined

distances are six, including the case whose individual Z-memory distance lies in $[6, 7]$. The weight-two subsystem implementations shown in the main-text defect figure instead have certified graphlike lower bounds and native fault witnesses attaining $d_c = d$. These statements refer to the specified gate order, routing, schedule, and fault set; all corresponding resource counts are unchanged.

G Extended Numerical Results

G.1 Evolution Dynamics

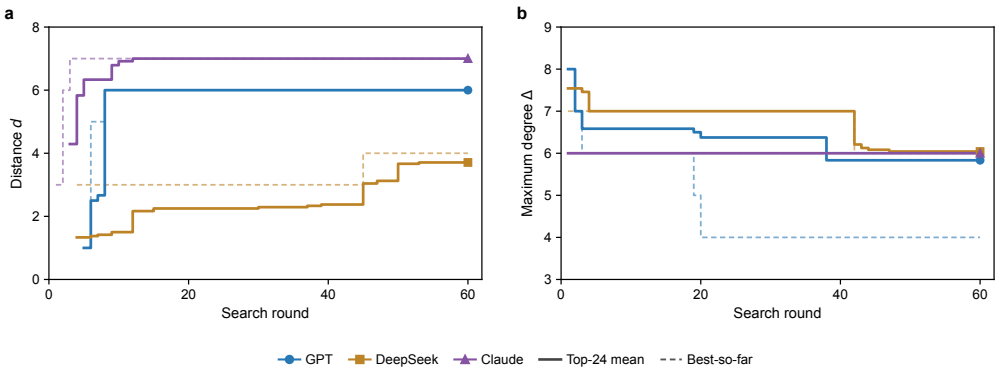

Figure 6: **Additional search histories. a:** Protection ($\Delta \leq 5$) and **b:** Connectivity ($d \geq 9$). Solid curves show the mean primary metric of the best 24 distinct metric profiles in the cumulative archive; dashed curves show the best-so-far value. The primary metric is d in **a**, where larger values are preferred, and Δ in **b**, where smaller values are preferred. The statistics retain each task's non-objective constraints and all original search-round indices. A top-24 curve starts once 24 profiles are available.

G.2 Circuit-level logical error rates

We evaluate the discovered $[\![25, 1, 16, 5]\!]$ subsystem implementation on the hardware instance with nine missing couplers. The circuit uses 25 data qubits and 22 measurement ancillas. We retain its compiled gate order and run separate X- and Z-basis memory experiments with $T = 5$ complete gauge-measurement cycles.

We use uniform circuit-level Pauli noise: each Hadamard gate and each idle location is followed by a single-qubit depolarizing channel of total probability p, and each CZ gate by a two-qubit depolarizing channel of total probability p. The nonidentity Pauli operators are equiprobable within each channel. Preparation errors and measurement-bit flips each occur with probability p. An idle location is an inactive occupied site in one scheduled circuit layer. This model includes idle noise in addition to the gate, preparation, and readout faults considered in the circuit-distance analysis.

We sample the noisy circuits using Stim (Gidney, 2021) and decode the full, undecomposed detector error model using belief propagation with ordered-statistics postprocessing (BP+OSD) (Roffe et al., 2020). We use parallel min-sum BP with a scaling factor of 0.625 and at most 50 iterations, followed by order-two combination-sweep OSD when BP does not converge. The parameters are fixed across all five error probabilities and both memory bases. We define $P_{\mathrm{L}} = N_{\mathrm{err}}/N$ as the fraction of complete memory experiments whose protected logical readout is incorrectly predicted. Each point uses $N = 10^5$ independent circuit shots, with no postselection.

G.3 A two-body gauge-measurement example

The GPT protection-task run yields a $[\![54, 2, 38, 6]\!]$ subsystem code with the following $G_X, G_Z \in \mathbb{F}_2^{45\times 54}$. Columns index data qubits and rows specify measured X- or Z-type gauge generators.

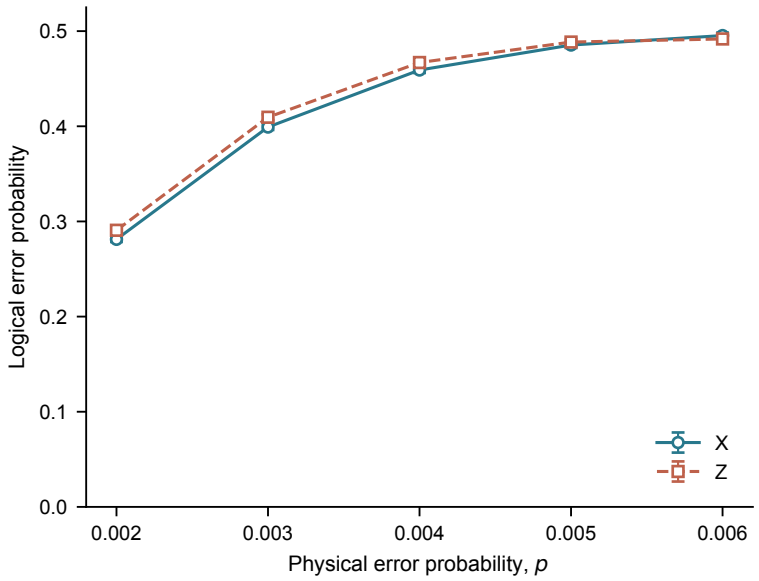

Figure 7: **Circuit-level logical memory failure probabilities for the discovered $[\![25, 1, 16, 5]\!]$ code under uniform Pauli noise.** Each curve contains the prescribed five physical error probabilities, $p = 0.002, 0.003, 0.004, 0.005, 0.006$. Points show failure probabilities after $T = 5$ measurement cycles; error bars are pointwise 95% Wilson binomial confidence intervals from 10^5 shots per point and basis. Connecting lines guide the eye.

Every row has weight two, so $w = 2$. All original measurement rows are retained; their number need not equal the matrix rank.

$$G_X = \begin{bmatrix}
01000000000000000000000000000000001000000000000000000000\\
10100000000000000000000000000000000000000000000000000000\\
00010001000000000000000000000000000000000000000000000000\\
00001100000000000000000000000000000000000000000000000000\\
00001000000000000000100000000000000000000000000000000000\\
00100010000000000000000000000000000000000000000000000000\\
00000001000100000000000000000000000000000000000000000000\\
00000000100010000000000000000000000000000000000000000000\\
00001000010000000000000000000000000000000000000000000000\\
00000010000100000000000000000000000000000000000000000000\\
00000000000001000000000000000000000000100000000000000000\\
00000000000011000000000000000000000000000000000000000000\\
00000000000000100000000000010000000000000000000000000000\\
00000000010000010000000000000000000000000000000000000000\\
00000000000010000000000100000000000000000000000000000000\\
00000000000010000010000000000000000000000000000000000000\\
00000000000000100000100000000000000000000000000000000000\\
00000000000000001100000000000000000000000000000000000000\\
00000000000000000000000000000000000010010000000000000000\\
00000000000000100000100000000000000000000000000000000000\\
00000000000000010000010000000000000000000000000000000000\\
00000000000000000011000000000000000000000000000000000000\\
00000000000000010000000000010000000000000000000000000000\\
00000000000000000000000000000000100000100000000000000000\\
00000000000000000000000000000000010000010000000000000000\\
00000000000000000000000000000000000011000000000000000000\\
00000000000000000000000000000000000000000000000000000000\\
00000000000000000000000000000000000000000000000000000000\\
00000000000000000000000000000000000000

$$G_Z = \begin{bmatrix}
1100000000000

Table 7: Three selected code-parameter profiles per model and setting after 60 rounds. The connectivity tasks maximize distance at $\Delta \leq 5$, maximize capacity at $\Delta \leq 6$, and minimize degree for $d \geq 9$, respectively.

Setting	Model	First	Second	Third
2 C	GPT	$[\![49, 1, 36, 7]\!]$	$[\![56, 1, 42, 7]\!]$	$[\![36, 1, 25, 6]\!]$
	DeepSeek	$[\![49, 1, 36, 7]\!]$	$[\![36, 1, 25, 6]\!]$	$[\![42, 1, 30, 6]\!]$
	Claude	$[\![49, 1, 36, 7]\!]$	$[\![36, 1, 25, 6]\!]$	$[\![25, 1, 0, 5]\!]$
4 C	GPT	$[\![49, 1, 36, 7]\!]$	$[\![49, 1, 24, 7]\!]$	$[\![49, 1, 16, 7]\!]$
	DeepSeek	$[\![49, 1, 36, 7]\!]$	$[\![36, 1, 25, 6]\!]$	$[\![42, 1, 30, 6]\!]$
	Claude	$[\![49, 1, 36, 7]\!]$	$[\![49, 1, 3, 7]\!]$	$[\![49, 1, 20, 7]\!]$
9 C	GPT	$[\![25, 1, 16, 5]\!]$	$[\![16, 1, 9, 4]\!]$	$[\![16, 1, 0, 4]\!]$
	DeepSeek	$[\![16, 1, 9, 4]\!]$	$[\![20, 1, 12, 4]\!]$	$[\![24, 1, 15, 4]\!]$
	Claude	$[\![36, 4, 0, 3]\!]$	$[\![36, 4, 16, 3]\!]$	$[\![38, 4, 0, 3]\!]$
2 Q + 2 C	GPT	$[\![36, 1, 25, 6]\!]$	$[\![42, 1, 30, 6]\!]$	$[\![42, 1, 31, 6]\!]$
	DeepSeek	$[\![36, 1, 25, 6]\!]$	$[\![42, 1, 30, 6]\!]$	$[\![48, 1, 35, 6]\!]$
	Claude	$[\![36, 1, 25, 6]\!]$	$[\![42, 1, 30, 6]\!]$	$[\![48, 1, 35, 6]\!]$
Protection	GPT	$[\![54, 2, 38, 6]\!]$	$[\![56, 2, 38, 6]\!]$	$[\![60, 2, 44, 6]\!]$
	DeepSeek	$[\![32, 2, 18, 4]\!]^{\dagger}$	$[\![26, 2, 0, 3]\!]^{\dagger}$	$[\![49, 2, 30, 3]\!]^{\dagger}$
	Claude	$[\![50, 2, 4, 7]\!]$	$[\![50, 2, 3, 7]\!]$	$[\![50, 2, 0, 7]\!]$
Capacity	GPT	$[\![75, 7, 52, 5]\!]$	$[\![75, 3, 48, 5]\!]$	$[\![50, 2, 32, 5]\!]$
	DeepSeek	$[\![75, 7, 52, 5]\!]$	$[\![75, 3, 48, 5]\!]$	$[\![50, 2, 32, 5]\!]$
	Claude	$[\![70, 10, 0, 5]\!]$	$[\![70, 9, 0, 5]\!]$	$[\![70, 8, 0, 5]\!]$
Connectivity	GPT	$[\![49, 1, 12, 9]\!]$	$[\![81, 1, 64, 9]\!]$	$[\![36, 1, 19, 4]\!]^{\dagger}$
	DeepSeek	$[\![63, 1, 4, 9]\!]$	$[\![81, 1, 64, 9]\!]$	$[\![50, 2, 0, 7]\!]^{\dagger}$
	Claude	$[\![61, 1, 0, 9]\!]$	$[\![63, 1, 4, 9]\!]$	$[\![81, 1, 64, 9]\!]$