QUANTUM ARCHITECTURE SEARCH FOR SYNDROME EXTRACTION IN FAULT-TOLERANT QUANTUM COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantum error correction (QEC) is a key enabler of fault-tolerant quantum computing, yet the strong code-level properties of modern quantum low-density parity-check (qLDPC) codes do not automatically translate into equally strong circuit-level performance. A central bottleneck lies in syndrome extraction, where CNOT ordering, global scheduling, and ancilla faults jointly determine error propagation and logical reliability. We introduce *syndrome-extraction architecture* (SEA) search, which jointly designs local CNOT orderings, global X/Z scheduling, and selective flag deployment. Focusing on Abelian two-block group-algebra (2BGA) codes, we exploit their group structure to share ordering templates across stabilizer measurements while retaining flexible flag placement. We develop a probabilistic quantum architecture search framework that samples candidate architectures, compiles them into valid global circuits using CP-SAT, and updates the architecture distribution from decoder-in-the-loop circuit-level feedback. Across 20 Abelian 2BGA codes, SEA search consistently achieves the lowest logical error rate among all baselines, including solver-based scheduling, fully flagged scheduling, and AlphaSyndrome. Moreover, the resulting circuits recover the code-level distance in 16 of the 20 evaluated instances. These results show that syndrome extraction is an architecture-level design problem and establish quantum architecture search as a practical methodology for fault-tolerant QEC circuit design.

1 INTRODUCTION

Quantum error correction (QEC) protects logical information from physical noise through redundant encoding and repeated syndrome measurements, making it central to large-scale fault-tolerant quantum computation (Campbell et al., 2017; Shor, 1995; Steane, 1996b; Terhal, 2015). High-rate quantum low-density parity-check (qLDPC) codes offer a promising route to reducing the physical resource overhead of fault tolerance (Breuckmann & Eberhardt, 2021; Panteleev & Kalachev, 2022; Xu et al., 2024). An important family comprises two-block group-algebra (2BGA) codes over Abelian groups (Lin & Pryadko, 2024; Shaw & Terhal, 2025). Its bivariate-bicycle (BB) subclass, constructed over a direct product of two cyclic groups, offers high thresholds and low qubit overhead in circuit-level simulations (Bravyi et al., 2024). Realizing these benefits requires addressing two challenges in syndrome extraction. First, data–ancilla CNOTs must be ordered and scheduled so that no qubit participates in two gates simultaneously (Zhang et al., 2026). Second, a single fault on a syndrome ancilla can propagate through subsequent CNOTs to multiple data qubits, producing a correlated *hook error* (Chamberland & Beverland, 2018). Such propagation can allow fewer physical faults to produce an undetected logical operation, reducing the circuit distance below the code distance (Beverland et al., 2024). Preserving a code's error-correcting capability during syndrome extraction is essential to translating its theoretical advantages into reliable circuit-level performance.

Existing automated circuit synthesis methods have mainly focused on CNOT ordering and parallel scheduling, using constraint solvers to reduce circuit depth (Yin et al., 2025; Zhang et al., 2026) or decoder-guided Monte Carlo tree search (MCTS) to reduce the logical error rate (LER) (Liu et al., 2026). Although suitable scheduling can mitigate the impact of hook errors, gate ordering alone does not generally guarantee that the circuit distance matches the code distance (Strikis et al., 2026). This limitation suggests that searching for better gate orderings within a fixed measurement structure may not be sufficient. Flagged measurements offer an auxiliary structure, using additional ancillas

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.

to record information about propagation faults and help the decoder distinguish error patterns that the stabilizer syndrome alone cannot resolve (Chamberland & Beverland, 2018; Chao & Reichardt, 2018; Pato et al., 2024). However, the additional coupling, preparation, and measurement operations also introduce noise, so the information gained from flags must be weighed against their physical cost. More importantly, the effectiveness of flag measurements depends on the CNOT ordering, while inserting flag gates changes the global scheduling requirements. A schedule optimized without flags therefore need not remain optimal once flags are introduced, motivating the joint design of CNOT scheduling and flag deployment. We thus ask the following question:

Can CNOT scheduling and flag deployment be jointly learned to produce more reliable syndrome-extraction architectures?

To address this question, we introduce syndrome-extraction architecture (SEA) search, applying quantum architecture search (QAS) (Du et al., 2022; Lu et al., 2023; Yan et al., 2026c) to the joint design of measurement circuits. For Abelian 2BGA codes, group-translated checks allow local CNOT orderings to be represented by shared templates. We search over subgroup partitions to choose which checks share a template, together with the template orderings and check-level flag deployment. A learned joint distribution generates these choices. Independent Bernoulli variables for individual checks allow both the number and locations of flag ancillas to vary, with separate decisions for the X and Z sectors. A compiler based on the CP-SAT constraint-programming solver translates each sampled design into a global schedule, allowing X/Z interleaving while preventing simultaneous operations on the same qubit and preserving the intended stabilizer measurements. Subject to these constraints, it seeks to minimize the number of data- and flag-CNOT layers. We evaluate the resulting circuits through quantum memory experiments and use their LERs after decoding to jointly update all probability parameters via score-function gradients.

We benchmark SEA search on 20 Abelian 2BGA codes spanning single-variable, bivariate, and trivariate bicycle constructions. Our method achieves the lowest LER on every benchmark against Auto-Stabilizer-Check (ASC) (Zhang et al., 2026), fully flagged ASC, and AlphaSyndrome (Liu et al., 2026). Relative to fully flagged ASC, the strongest baseline, it reduces LER by 5.2–35.7%. The resulting circuits recover the code-level distance on 16 of the 20 benchmarks. The architectures achieving these gains exhibit code-dependent choices of template sharing and flag deployment. Intermediate subgroup partitions are selected on eleven benchmarks, while the remaining codes use global sharing or check-specific orderings. Partial flag coverage is selected on 19 benchmarks, reducing flag-ancilla use by up to 44.0% compared with fully flagged ASC. Ablations on five codes further show that learning ordering distributions and flag deployment yields lower LER than random ordering or fixed zero or full flag coverage. These results establish joint architecture search as an approach to translating algebraic code structure into circuits with improved logical reliability.

2 PRELIMINARIES

Stabilizer and CSS codes. An $[\![n, k, d]\!]$ stabilizer code encodes k logical qubits into n physical qubits. Its codespace is the simultaneous $+1$ eigenspace of an Abelian Pauli subgroup $\mathcal{S}$ with $-I \notin \mathcal{S}$. The code distance d is the minimum Pauli weight of a nontrivial logical operator, with global phases ignored (Terhal, 2015; Gottesman, 1997). A Calderbank–Shor–Steane (CSS) code (Calderbank & Shor, 1996; Steane, 1996a) has separate X- and Z-type checks, represented by binary support matrices $H_X \in \mathbb{F}_2^{m_X \times n}$ and $H_Z \in \mathbb{F}_2^{m_Z \times n}$ satisfying

$$H_X H_Z^\top = 0, \qquad k = n - \operatorname{rank} H_X - \operatorname{rank} H_Z. \tag{1}$$

All binary matrix operations are over $\mathbb{F}_2$. The syndrome records whether an error commutes or anticommutes with each stabilizer check. Errors with the same syndrome may differ by a nontrivial logical operator, so decoding identifies a logical error class rather than a unique physical error (Gottesman, 2009; Roffe, 2019).

Abelian 2BGA codes (Shaw & Terhal, 2025). Let $G = C_{\ell_1} \times \cdots \times C_{\ell_t}$ be a finite Abelian group of order $N = \prod_{j=1}^{t} \ell_j$, written additively. For a support set $\mathcal{U} \subseteq G$, define the translation matrix $T(\mathcal{U})$ by $[T(\mathcal{U})]_{g,h} = \mathbf{1}[h - g \in \mathcal{U}]$. Given supports $\mathcal{A}, \mathcal{B} \subseteq G$, set $A = T(\mathcal{A})$ and $B = T(\mathcal{B})$. The corresponding two-block group-algebra code has check matrices

$$H_X = [A \mid B], \qquad H_Z = [B^\top \mid A^\top]. \tag{2}$$

Since $AB = BA$, the CSS commutation condition holds (Lin & Pryadko, 2024; Panteleev & Kalachev, 2021a; Voss et al., 2025). The code has $n = 2N$ data qubits in two group-indexed

blocks, with $m_X = m_Z = N$. Each check has weight $w = |\mathcal{A}| + |\mathcal{B}|$. Checks within a sector are translations of a common support pattern.

Example 1 (From polynomials to check matrices)**.** Consider $G = C_6 = \{0, 1, 2, 3, 4, 5\}$, with addition modulo 6. Let x denote the cyclic generator, so that $x^6 = 1$. Choose the group-algebra polynomials $1 + x^2$ and $1 + x$, whose supports are $\mathcal{A} = \{0, 2\}$ and $\mathcal{B} = \{0, 1\}$, respectively. A support set therefore records the displacements associated with nonzero polynomial coefficients. Figure 1 illustrates the Tanner graph and a corresponding syndrome-extraction circuit for this code.

In $T(\mathcal{U})$, the row index g identifies a check, and the column index h identifies a data-qubit position within one block. The entry is 1 when the relative displacement $h - g$ belongs to $\mathcal{U}$. With rows and columns ordered as $0, 1, 2, 3, 4, 5$, the unit translation is represented by

$$P = T(\{1\}) = \begin{pmatrix} 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1\\ 1&0&0&0&0&0 \end{pmatrix}, \qquad P^6 = I_6. \tag{3}$$

Replacing each monomial x^u by P^u gives $A = T(\mathcal{A}) = T(\{0\}) + T(\{2\}) = I_6 + P^2, B = T(\mathcal{B}) = I_6 + P$, where matrix addition is over $\mathbb{F}_2$. Concatenating the two blocks yields

$$H_X = [A \mid B] = \left[\begin{array}{cccccc|cccccc} 1&0&1&0&0&0&1&1&0&0&0&0\\ 0&1&0&1&0&0&0&1&1&0&0&0\\ 0&0&1&0&1&0&0&0&1&1&0&0\\ 0&0&0&1&0&1&0&0&0&1&1&0\\ 1&0&0&0&1&0&0&0&0&0&1&1\\ 0&1&0&0&0&1&1&0&0&0&0&1 \end{array}\right]. \tag{4}$$

The columns correspond to the data qubits $L_0, \ldots, L_5, R_0, \ldots, R_5$. In the Tanner graph shown in Fig. 1(a), each nonzero matrix entry connects a check to a data qubit in its support. For example, row $g = 0$ specifies the check $X_{L_0} X_{L_2} X_{R_0} X_{R_1}$. Row $g = 1$ shifts every data-qubit index by one modulo 6, giving $X_{L_1} X_{L_3} X_{R_1} X_{R_2}$. The Z-check matrix is $H_Z = [B^\top \mid A^\top]$. Since A and B are polynomials in the same matrix P, they commute and satisfy $H_X H_Z^\top = 0$. The construction therefore gives a cyclic 2BGA code on $n = 12$ data qubits, with $m_X = m_Z = 6$ and check weight 4.

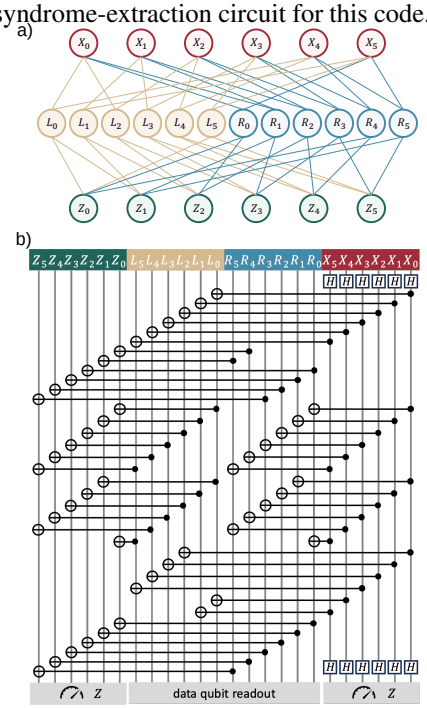

Figure 1: **From code-level to circuit-level. a)** Tanner graph of the cyclic 2BGA code in Example 1. **b)** A corresponding measurement circuit, where each check is assigned a dedicated syndrome ancilla.

Syndrome extraction and flags. Each check is measured using a dedicated syndrome ancilla (Fig. 1(b)). For an X-type check, the ancilla is prepared in $|+\rangle$, used as the control of CNOTs to the data qubits in the check support, and measured in the X basis. For a Z-type check, the ancilla is prepared in $|0\rangle$, used as the CNOT target, and measured in the Z basis (Fowler et al., 2012). A fault with a propagating ancilla component can spread through subsequent CNOTs to several data qubits, producing a correlated *hook error*. Its support depends on the CNOT ordering (Yoder & Kim, 2017; Beverland et al., 2024). Flag ancillas for X- and Z-type checks are denoted by f_X and f_Z, respectively. Their outcomes signal propagation faults and are combined with the syndrome record to distinguish otherwise ambiguous error patterns (Chamberland & Beverland, 2018; Chao & Reichardt, 2018; Pato et al., 2024). See Appendix A for circuit and propagation details.

Circuit-level distance. Circuit faults can affect both the data and the syndrome readout. A detector tests a parity of measurement outcomes whose value is deterministic in the absence of faults. A detection event occurs when this parity differs from its ideal value (Gidney, 2021). Comparing to code distance d, both distances characterize undetected logical errors. But d counts data-qubit Pauli weight, whereas d_{circ} counts physical faults in the extraction circuit.

Definition 1 (Circuit distance)**.** For a fixed circuit, fault model, and measurement protocol, the circuit distance d_{circ} is the minimum number of physical faults that jointly leave all detectors unchanged while flipping at least one logical observable (Beverland et al., 2024; Derks et al., 2025).

Example 2 (Distance loss in the Gross code (Strikis et al., 2026))**.** The Gross code is a $[\![144, 12, 12]\!]$ BB code (Bravyi et al., 2024). Strikis et al. (2026) analyze non-interleaved CNOT-based extraction

with one syndrome ancilla per check and no flags. They prove $d_{\mathrm{circ}} \leq 10$ when each sector uses one single translated local ordering template. Even allowing independent orderings for all checks leaves $d_{\mathrm{circ}} \leq 11 < d = 12$.

3 METHODS

Figure 2 presents our framework for syndrome-extraction architecture search. Sec. 3.1 defines the architecture space and subgroup-based sharing of local CNOT orderings for Abelian 2BGA codes. Sec. 3.2 introduces the probabilistic search procedure, which generates candidate designs, compiles their global schedules, and updates the architecture distribution using circuit-level LER feedback.

3.1 SYNDROME-EXTRACTION ARCHITECTURE

Architecture representation. Fixed stabilizer checks specify the observables to be measured but leave freedom in their measurement circuits. Motivated by Example 2, we consider three design choices for circuit-level syndrome extraction: local CNOT orderings, global scheduling with X/Z interleaving, and flag deployment. We define the following architecture, illustrated in Fig. 2 (top).

Definition 2 (Syndrome-extraction architecture (SEA))**.** For general CSS check matrices H_X, H_Z, a dedicated syndrome ancilla per check, and a prescribed flag gadget, a SEA is a triple

$$\gamma = (\pi, \mathbf{M}, \tau), \tag{5}$$

where π specifies the local data-CNOT orderings, $\mathbf{M}$ the flag-deployment mask, and τ the execution ticks of all data and flag CNOTs.

Index the checks in H_X and H_Z jointly by $i = 1, \ldots, m_X + m_Z$, with syndrome ancilla a_i assigned to check i. For a check of weight w_i, π_i lists each data qubit in its support exactly once; $\pi_i(j)$ denotes the qubit visited in the j-th data CNOT with a_i, for $j = 1, \ldots, w_i$. The mask $\mathbf{M} \in \{0,1\}^{m_X+m_Z}$ specifies flag deployment: when $M_i = 1$, a dedicated flag ancilla f_i is coupled to a_i using the prescribed gadget; otherwise, the check is unflagged. The total number of flag ancillas is $\sum_{i=1}^{m_X+m_Z} M_i$.

The global schedule τ assigns execution ticks to all data and flag CNOTs. We write $\tau_i(j)$ for the tick of the j-th data CNOT of check i, with $\tau_i(j) < \tau_i(j+1)$ for $j = 1, \ldots, w_i - 1$. The flag CNOTs are inserted according to the prescribed gadget while preserving the data-CNOT ordering. A valid architecture implements the intended stabilizer measurements and permits simultaneous CNOTs only on disjoint qubit pairs, including data, syndrome, and flag qubits. When X- and Z-check measurements are interleaved, their relative gate ordering over shared data qubits must additionally satisfy the even-inversion condition, ensuring that the resulting circuit remains equivalent to the intended joint stabilizer measurement (Zhang et al., 2026; Viszlai et al., 2026).

Subgroup-based template sharing. The general SEA definition allows each syndrome ancilla to have an independent local ordering π_i. For Abelian 2BGA codes, however, checks within the same Pauli sector are group-translated copies of a common local interaction pattern (Lin & Pryadko, 2024; Shaw & Terhal, 2025). We exploit this structure to share ordering templates across multiple checks, reducing the number of independently searched orderings. Choose a subgroup $H \leq G$, which induces the coset decomposition

$$G = \bigsqcup_{c=1}^{C} (g_c + H), \qquad C = [G : H] = \frac{|G|}{|H|}. \tag{6}$$

Here g_c is a representative of the c-th coset. We assign one ordering template $\bar{\pi}_c^{\alpha}$ to each coset c and Pauli sector $\alpha \in \{X, Z\}$, giving $2C$ templates for the $2N$ checks. Within each sector, all checks in a coset use the same sequence of relative data interactions. Each interaction is represented by (β, u), where $\beta \in \{L, R\}$ denotes the data block and $u \in G$ the corresponding group displacement. For an X check, the available interactions are (L, u) with $u \in \mathcal{A}$ and (R, u) with $u \in \mathcal{B}$; for a Z check, they are (L, u) with $u \in -\mathcal{B}$ and (R, u) with $u \in -\mathcal{A}$. Here $-\mathcal{U} = \{-u : u \in \mathcal{U}\}$.

Let $g_i \in G$ be the group index of check i in coset c. If $\bar{\pi}_c^{\alpha}(j) = (\beta, u)$, the corresponding check-specific ordering is $\pi_i(j) = \beta_{g_i+u}$, with addition performed in G. Thus, checks in the same coset use the same relative CNOT sequence on translated physical data qubits. Under this parameterization, an Abelian 2BGA architecture is written as

$$\widetilde{\gamma} = (H, \bar{\pi}, \mathbf{M}, \tau), \qquad \bar{\pi} = \{\bar{\pi}_c^{\alpha} : c = 1, \ldots, C,\ \alpha \in \{X, Z\}\}, \tag{7}$$

where H and $\bar{\pi}$ induce the check-specific orderings π in Definition 2, while τ and $\mathbf{M}$ retain their original meanings. The two extreme choices recover familiar settings: $H = G$ gives one order-

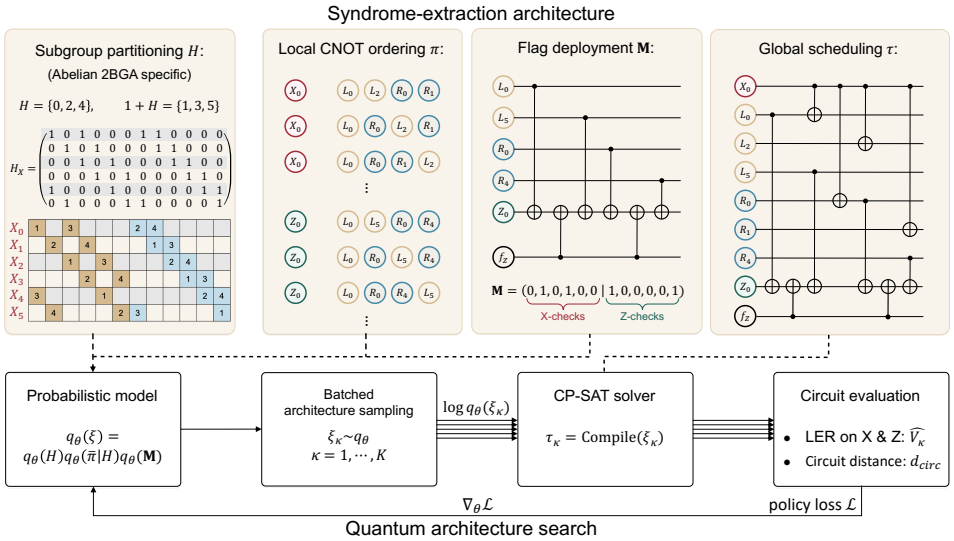

Figure 2: **Syndrome-extraction architecture and probabilistic search. Top:** Components of a grouped syndrome-extraction architecture for the code in Example 1. The subgroup H partitions the checks into groups that share local CNOT ordering templates. The orderings π specify data-qubit access sequences, the global schedule τ assigns execution times, and the mask $\mathbf{M}$ specifies flag deployment for individual syndrome ancillas. **Bottom:** A trainable probabilistic model $q_\theta(\xi)$ generates batches of designs $\xi = (H, \bar{\pi}, \mathbf{M})$, where $\bar{\pi}$ collects the shared ordering templates. CP-SAT constructs a global schedule for each sampled design. Circuit evaluation estimates the memory-X/Z logical error rates used as search costs. Circuit distance is assessed for the selected circuits. The estimated LER costs $\widehat{V}_\kappa$ and sampled log-probabilities $\log q_\theta(\xi_\kappa)$ are used to estimate $\nabla_\theta \mathcal{L}$ for updating the probabilistic model.

ing template per Pauli sector, as in globally shared single-template extraction (Strikis et al., 2026; Bravyi et al., 2024), whereas $H = \{0\}$ recovers fully check-specific orderings. Intermediate subgroups interpolate between these cases. The sharing applies only to local data-CNOT orderings; flag assignments and global execution times remain check-specific.

Example 3 (Subgroup template sharing)**.** For the C_6 code in Example 1 and Fig. 2, choose $H = \{0, 2, 4\}$, giving the two cosets $H = \{0, 2, 4\}$ and $1 + H = \{1, 3, 5\}$. The X checks indexed by $g_i \in H$ share template $\bar{\pi}_1^X$, and those indexed by $g_i \in 1 + H$ share template $\bar{\pi}_2^X$. Since $\mathcal{A} = \{0, 2\}$ and $\mathcal{B} = \{0, 1\}$, each template orders the four interaction labels $(L, 0), (L, 2), (R, 0), (R, 1)$. For $\bar{\pi}_1^X = ((L, 0), (R, 0), (L, 2), (R, 1))$ in Fig. 2, the third entry $(L, 2)$ maps to L_2, L_4, L_0 data qubits for $g \in H$. The corresponding indices are $0 + 2$, $2 + 2$, and $4 + 2 \equiv 0 \pmod 6$.

3.2 Probabilistic architecture search

For a given Abelian 2BGA code, we search over the grouped SEA defined in equation 7. Directly sampling the global schedule τ is undesirable, because it must satisfy hard circuit-validity constraints, including qubit exclusivity and X/Z measurement compatibility. We therefore separate architecture generation from global scheduling. The probabilistic model samples the subgroup, local ordering templates, and flag deployment, while a constraint-aware compiler constructs a valid global schedule for each sampled design. The resulting circuit-level performance is then used to update the architecture distribution, as illustrated in Fig. 2 (bottom).

Probabilistic architecture generation. We denote the sampled architectural choices by $\xi = (H, \bar{\pi}, \mathbf{M})$, which are generated by the probabilistic model. We parameterize their distribution as

$$q_\theta(\xi) = q_\theta(H)\, q_\theta(\bar{\pi} \mid H)\, q_\theta(\mathbf{M}), \tag{8}$$

where θ denotes the trainable probability parameters. The factorization in Eq. (8) reflects the structure of the architecture space. We first sample the subgroup H from a learned categorical distribution over a prescribed set of candidate subgroups. Once H is fixed, its cosets determine the number and membership of the ordering-template groups. For each coset c and Pauli sector $\alpha \in \{X, Z\}$, we then sample a template $\bar{\pi}_c^\alpha$. Each template is generated sequentially: at every position, trainable logits define a categorical distribution over the remaining relative interactions, from which the next

interaction is sampled and subsequently removed. This produces a valid ordering without explicitly enumerating all possible permutations.

Flag deployment is parameterized independently of the subgroup partition and ordering templates. Each syndrome ancilla is associated with a Bernoulli variable $M_i \in \{0, 1\}$, whose learned probability determines whether a flag ancilla is deployed for check i. The parameters θ therefore collect the logits governing subgroup selection, template generation, and per-check flag deployment. At each search iteration, we draw a batch $\{\xi_\kappa\}_{\kappa=1}^{K}$ independently from q_θ, with $\xi_\kappa = (H_\kappa, \bar{\pi}_\kappa, \mathbf{M}_\kappa)$, and retain the corresponding log-probabilities $\log q_\theta(\xi_\kappa)$. These log-probabilities are used in the subsequent policy-gradient update. The sampled designs are then passed to the subsequent compiler.

Constraint-aware circuit compilation. Each sampled design $\xi_\kappa = (H_\kappa, \bar{\pi}_\kappa, \mathbf{M}_\kappa)$ is compiled independently into a valid syndrome-extraction circuit. The sampled subgroup and templates first determine the check-specific data-CNOT orderings, while the flag mask specifies which syndrome ancillas are augmented with the prescribed flag gadget. For each flagged check, two syndrome–flag CNOTs are inserted after the first and before the last data CNOT, forming the maximal flag window used throughout this work.

Given these local interaction sequences, CP-SAT (Perron et al., 2023) constructs the global schedule

$$\tau_\kappa = \mathrm{Compile}(\xi_\kappa). \tag{9}$$

The compiler assigns an execution tick to every data and flag CNOT while preserving each check's ordering and enforcing qubit exclusivity, so no data, syndrome, or flag qubit participates in more than one CNOT at the same tick. The schedule may interleave the CNOT sequences of X and Z checks. To ensure that interleaving implements the intended joint stabilizer measurement, every overlapping X–Z check pair must satisfy the even-inversion condition (Zhang et al., 2026; Viszlai et al., 2026): the relative ordering of their interactions on shared data qubits must have even parity.

Subject to these validity constraints, CP-SAT minimizes the total number of CNOT layers, including both data and flag interactions. The resulting schedule τ_κ completes the grouped architecture $\widetilde{\gamma}_\kappa = (H_\kappa, \bar{\pi}_\kappa, \mathbf{M}_\kappa, \tau_\kappa)$, which is then passed to the circuit-level evaluator. The complete constraint encoding and solver settings are given in Appendix C.

Decoder-in-the-loop optimization. After compilation, each architecture $\widetilde{\gamma}_\kappa$ is evaluated using memory-X and memory-Z experiments under a fixed noise model and decoding protocol. The decoder takes both syndrome and flag measurement records as input. Following the reporting convention of Wang et al. (2026), we report the effective logical error rate per logical qubit per syndrome-extraction round. For each memory experiment, the block logical failure probability is first normalized per logical qubit and then converted to a per-round rate following Bausch et al. (2024); the complete normalization is given in Appendix E. Let $\widehat{P}^X_{\mathrm{L},\kappa}$ and $\widehat{P}^Z_{\mathrm{L},\kappa}$ denote the resulting estimates for candidate κ. We use their average, $\widehat{V}_\kappa = (\widehat{P}^X_{\mathrm{L},\kappa} + \widehat{P}^Z_{\mathrm{L},\kappa})/2$, as the scalar feedback for the architecture search. This evaluation naturally accounts for both the additional information provided by flag measurements and the noise introduced by their extra circuit operations.

The probabilistic model is optimized to minimize the expected circuit-level cost, $\mathcal{L}(\theta) = \mathbb{E}_{\xi\sim q_\theta}[V(\xi)]$, where $V(\xi)$ denotes the expected cost of the circuit obtained after compiling and evaluating design ξ. For a sampled batch, we use the mean cost of the other candidates, $\overline{V}_{-\kappa} = (K-1)^{-1}\sum_{j\neq\kappa}\widehat{V}_j$, as a leave-one-out baseline. The score-function estimator (Williams, 1992; Ahmadian et al., 2024; Zhang et al., 2022) then gives

$$\nabla_\theta \mathcal{L}(\theta) \approx \frac{1}{K}\sum_{\kappa=1}^{K}\left(\widehat{V}_\kappa - \overline{V}_{-\kappa}\right)\nabla_\theta \log q_\theta(\xi_\kappa). \tag{10}$$

The circuit costs and baselines are treated as constants in this update, so gradients propagate only through the sampled log-probabilities retained during architecture generation, without differentiating through either CP-SAT or the circuit evaluator. Thus, subgroup selection, shared ordering templates, and per-check flag probabilities are updated jointly from circuit-level decoding feedback.

Circuit distance and architecture characteristics are evaluated for selected circuits rather than used as optimization objectives. Optimization settings and candidate selection are detailed in Appendix D. Memory protocols, decoding settings, and logical-error normalization are provided in Appendix E.

Table 1: Circuit performance across 20 Abelian 2BGA CSS codes. All LER values are in units of 10^{-3}. C is the selected group count. F_X and F_Z denote the numbers of flag ancillas used for X and Z checks, respectively; entries are displayed as used/available.

		Our architecture			Ours		ASC		ASC+flag		AlphaSyndrome	
$[\![n, k, d]\!]$	w	C	F_X	F_Z	LER	d_{circ}	LER	d_{circ}	LER	d_{circ}	LER	d_{circ}
Single-variable bicycle codes (SB)												
$[\![60, 4, 8]\!]$	5	1	19/30	23/30	0.803	8	1.210	6	1.140	8	1.560	5
$[\![54, 4, 8]\!]$	6	3	24/27	24/27	0.871	7	2.210	5	1.310	7	8.500	4
$[\![60, 4, 9]\!]$	7	2	26/30	29/30	0.539	9	1.040	7	0.803	8	12.500	5
$[\![84, 6, 10]\!]$	7	1	39/42	35/42	0.259	10	0.334	10	0.292	10	2.690	8
$[\![42, 2, 7]\!]$	8	21	16/21	20/21	2.380	5	4.170	4	3.540	5	24.900	3
IBM bivariate bicycle codes												
$[\![72, 12, 6]\!]$	6	12	33/36	32/36	1.170	6	1.480	4	1.440	6	3.230	4
$[\![90, 8, 10]\!]$	6	3	44/45	43/45	0.225	10	0.439	7	0.244	10	0.957	7
$[\![108, 8, 10]\!]$	6	1	47/54	51/54	0.106	10	0.163	8	0.125	10	0.704	9
$[\![144, 12, 12]\!]$	6	1	54/72	55/72	0.071	12	0.146	10	0.075	12	0.787	10
General bivariate bicycle codes												
$[\![56, 6, 8]\!]$	6	1	28/28	28/28	0.762	8	0.955	5	0.804	7	2.640	5
$[\![48, 4, 7]\!]$	7	4	16/24	14/24	0.906	7	1.580	6	1.060	7	5.610	4
$[\![72, 4, 8]\!]$	7	4	28/36	29/36	0.601	8	0.916	6	0.752	8	4.390	4
$[\![96, 8, 10]\!]$	7	8	32/48	26/48	0.100	10	0.200	10	0.106	10	1.420	8
$[\![32, 2, 6]\!]$	8	16	14/16	14/16	4.440	6	7.490	4	5.080	5	21.500	3
$[\![48, 2, 8]\!]$	8	1	24/24	22/24	1.530	8	2.710	7	1.730	8	16.700	4
$[\![64, 2, 10]\!]$	8	8	26/32	30/32	0.826	10	1.980	8	0.877	9	24.700	7
Trivariate bicycle codes (TB)												
$[\![60, 4, 8]\!]$	7	2	20/30	17/30	0.635	8	0.823	7	0.765	7	4.630	6
$[\![84, 6, 10]\!]$	6	2	25/42	22/42	0.165	9	0.424	7	0.257	9	1.880	7
$[\![90, 8, 9]\!]$	7	9	44/45	42/45	0.345	9	0.534	9	0.471	9	2.450	5
$[\![128, 20, 8]\!]$	8	1	57/64	63/64	0.438	7	0.850	5	0.591	8	5.730	4

4 Numerical results

4.1 Experimental setup and evaluation metrics

Benchmarks and baselines. We evaluate our method on 20 Abelian 2BGA CSS codes (Lin & Pryadko, 2024; Bravyi et al., 2024; Voss et al., 2025), with their constructions and parameters provided in Appendix F.1. We adopt two state-of-the-art methods for automated syndrome-extraction circuit design as baselines: ASC (Zhang et al., 2026) and AlphaSyndrome (Liu et al., 2026). ASC uses constraint-based circuit compilation, whereas AlphaSyndrome uses decoder-guided Monte Carlo tree search. For ASC, we consider both its original unflagged circuits and our fully flagged extension. For AlphaSyndrome, we use the authors' unflagged search framework with a modified reward. Baseline implementation details are provided in Appendix F.2.

Evaluation. We evaluate LER under the SI1000 noise model (Gidney et al., 2021) at $p = 0.003$, using a common configuration of the Telescoping decoder (Bhardwaj et al., 2026). For final comparisons, we hold each method's selected circuits fixed and evaluate them using fresh Monte Carlo samples independent of those used for search or candidate selection. We also report upper bounds on the circuit-level distance d_{circ}, obtained by searching for undetectable logical fault configurations using randomized information-set search (Pryadko et al., 2022). The LER and distance evaluation protocols are detailed in Appendix E. For our architectures, we additionally report the selected group count C and the numbers of flag ancillas used for X and Z checks. We compare the CNOT depths of the selected circuits to assess depth–performance tradeoffs (Appendix G.1).

Search settings. Our method performs 40 optimization updates per code, evaluating 128 candidate circuits at each update. Search feedback is obtained from memory-X and memory-Z experiments with two rounds of syndrome extraction. Sampling in each memory basis stops at 30 decoding failures or 10^6 shots, whichever occurs first. Candidate selection from the recorded search evaluations is described in Appendix D.2. Optimizer settings and initialization are provided in Appendix D, and compilation details in Appendix C.

4.2 Circuit performance across benchmark codes

Tab. 1 summarizes the performance across the 20 benchmark codes. Our method achieves the lowest LER on all benchmarks. ASC+flag provides the strongest baseline throughout the benchmark set,

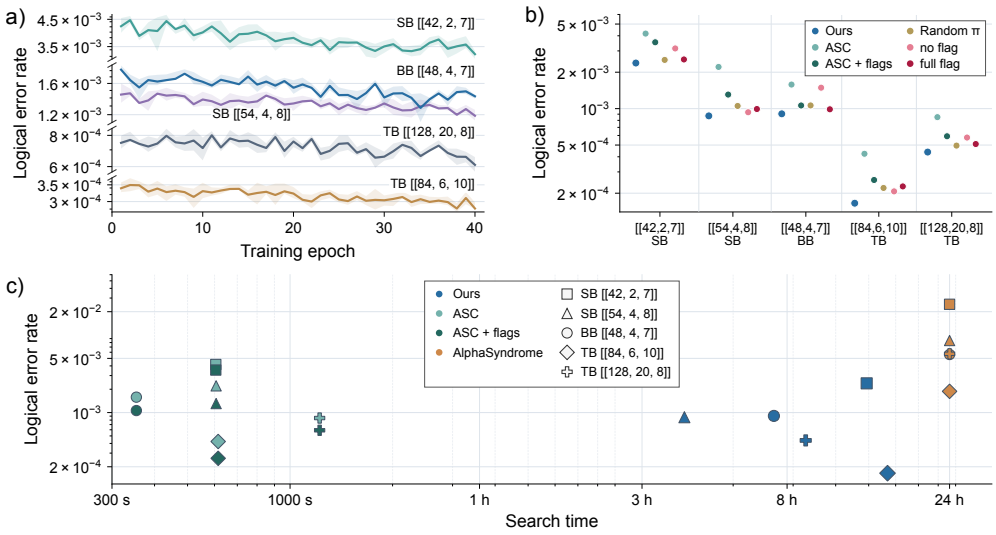

Figure 3: **Learning dynamics, ablations, and search efficiency.** **a)** Mean LER of the ten best candidates in each training batch over 40 optimization updates. Shaded regions show the range from the lowest to the highest LER among these ten candidates. **b)** LER of the full method, three ablation variants (Random π, no flags, and full flags), and the ASC and ASC+flag baselines. **c)** LER versus search time for our method, ASC, ASC+flag, and AlphaSyndrome. Each point represents one method applied to one code. Time covers candidate generation for ASC and ASC+flag, and search with feedback evaluation for our method and AlphaSyndrome.

and our method reduces LER by 5.2–35.7% relative to this baseline. For example, on the $[\![60, 4, 9]\!]$ SB code with check weight seven, the LER decreases from 8.03×10^{-4} to 5.39×10^{-4}. Relative to unflagged ASC and AlphaSyndrome, the reductions are 20.2–61.0% and 48.5–96.7%, respectively.

The circuit-level distance equals the code distance on 16 of the 20 benchmarks. Relative to unflagged ASC, our circuits attain higher circuit-level distances on 17 benchmarks and the same distances on the remaining three. For the $[\![64, 2, 10]\!]$ BB code, our circuit attains $d_{\text{circ}} = 10$, compared with eight for ASC and nine for ASC+flag. Circuit distance and LER provide complementary characterizations (Derks et al., 2025). For the $[\![128, 20, 8]\!]$ TB code, our circuit has $d_{\text{circ}} = 7$, compared with eight for ASC+flag, yet achieves a lower LER of 4.38×10^{-4} versus 5.91×10^{-4}. Thus, the circuit with the larger distance need not have the lower LER at the evaluated physical error rate.

We next examine the architectures underlying these performance gains. The subgroup choices in Tab. 1 span different degrees of template sharing. Eleven benchmarks use intermediate subgroup partitions ($1 < C < |G|$), seven use global sharing within each sector ($C = 1$), and two use check-specific orderings ($C = |G|$). Even among the IBM BB codes with check weight six, the selected group count ranges from one to twelve. These results reveal code-dependent choices of sharing granularity within the same search framework.

Partial flag coverage is selected on 19 benchmarks, and the numbers of flagged X and Z checks differ on 17 benchmarks. For the $[\![60, 4, 8]\!]$ TB code with check weight seven, the selected circuit uses 37 flag ancillas instead of the 60 required by ASC+flag. This reduces flag use by 38.3% while lowering LER by 17.0%. The largest saving occurs for the $[\![84, 6, 10]\!]$ TB code with check weight six, which uses 47 rather than 84 flag ancillas, a reduction of 44.0%.

We further examine how candidate quality evolves during optimization, how restricting ordering and flag deployment affects LER, and how much computation the search requires. Figure 3 presents these analyses on five benchmark codes.

Learning dynamics. As shown in Fig. 3(a), the mean LER of the ten best candidates in each batch decreases by 17.0–22.7% from the first to the fortieth update across the five codes. All five curves attain their lowest values between the 34th and 40th updates, showing that improved candidates continue to emerge late in the search. Batch-to-batch fluctuations remain visible throughout training, consistent with stochastic architecture sampling and finite-shot evaluation. We extend training to 100 updates on four codes to assess the improvement available beyond the main search budget (Appendix G.2).

Ablation study. We next investigate the contributions of ordering optimization and selective flag deployment through three ablation variants. In the Random π variant, each ordering template $\bar{\pi}_c^{\alpha}$ is sampled uniformly from all permutations for every candidate. Subgroup selection and flag deployment remain learned, and checks within each group retain template sharing. The no-flags and full-flags variants fix the flag mask to $\mathbf{M} = \mathbf{0}$ and $\mathbf{M} = \mathbf{1}$, respectively, while retaining our subgroup and ordering optimization. Detailed configurations are provided in Appendix G.3.

Figure 3(b) compares these variants with the full method, alongside ASC and ASC+flag. AlphaSyndrome is omitted from this panel for visual clarity owing to its substantially higher LERs; its results appear in Tab. 1 and Fig. 3(c). The full method achieves lower LER than all three variants on all five codes. Compared with Random π, it reduces LER by 5.7–25.2%, supporting the benefit of learning the ordering distributions. The relative performance of zero and full flag coverage depends on the code. Full coverage performs better on three codes, but worse on the $[\![54, 4, 8]\!]$ SB and $[\![84, 6, 10]\!]$ TB codes. For the $[\![84, 6, 10]\!]$ TB code, the full method selects 47 of the 84 available flag ancillas and achieves an LER of 1.65×10^{-4}. The corresponding LERs are 2.07×10^{-4} without flags and 2.28×10^{-4} with full coverage. This comparison supports retaining flag deployment as a learned design choice alongside subgroup selection and ordering optimization.

Search efficiency. Figure 3(c) compares LERs with the computational time used by each method. Our method takes 4.00–15.78 hours across the five codes, including circuit compilation and feedback evaluation. ASC and ASC+flag generate their candidate circuits in approximately 5.9–20.4 minutes, excluding the LER evaluations used for candidate selection. Our method achieves lower LER than both ASC variants on every code. Compared with AlphaSyndrome, whose search time includes feedback evaluation, our method achieves lower LER with shorter search times on all five codes. Computational settings and timing details are provided in Appendix G.1. To examine performance away from the search noise level, we evaluate the fixed circuits selected at $p = 0.003$ over physical error rates from 0.001 to 0.005 on five benchmark codes. The results in Appendix G show that the LER advantage persists across the tested range, with smaller margins at the highest error rate.

5 Discussion and Limitations

Our results suggest that high-quality syndrome extraction should be treated as a joint architecture-design problem rather than CNOT scheduling alone. Across evaluated codes, our framework jointly searches subgroup-based template sharing, local CNOT orderings, and flag deployment, while the constraint-aware compiler constructs valid global schedules. The resulting circuits achieve lower LERs than those obtained by optimizing the schedule alone or applying flags uniformly. This highlights a feature of QEC circuit design: useful transformations are often code dependent. Exploiting the algebraic structure of the code to constrain and organize the search can be as important as the search algorithm. Fault-tolerant circuit optimization benefits from combining generic optimization tools with QEC-specific knowledge of error propagation and code structure.

This work focuses on Abelian 2BGA codes, where the group structure provides a particularly natural mechanism for sharing local ordering templates. A direct extension is to investigate non-Abelian 2BGA codes, for which analogous group structure remains available but the resulting template parameterization is less straightforward. The general SEA formulation itself is not restricted to 2BGA codes and can also be applied to broader qLDPC families such as lifted-product and hypergraph-product codes. In these settings, however, the subgroup-based sharing used here may no longer be available, and the search would reduce to a more general architecture-optimization problem with check-specific local orderings. Developing alternative structural parameterizations for these code families is an interesting direction for future work.

Limitations. The main practical limitation of the current approach is search cost. Our optimization uses circuit-level logical error rates as feedback, and accurately estimating a low LER can require a large number of Monte Carlo shots for every sampled architecture. The resulting decoder calls dominate the search time and make direct scaling to substantially larger codes increasingly expensive. One promising direction is to exploit transfer within a code family (Yan et al., 2026b): structural choices such as subgroup partitions or template-sharing patterns learned on smaller instances may provide useful initializations or restricted search spaces for larger codes. Moreover, improvements in circuit-level decoding speed (Yan et al., 2026a) would likewise translate directly into faster architecture search.

AI USE STATEMENT

Generative AI tools were used to assist with the preparation and revision of the manuscript, including language polishing, refinement of academic expression, and improvements to clarity and readability. The authors are responsible for ensuring that AI-assisted revisions accurately reflect the technical content and intended meaning of the work. The authors take full responsibility for the accuracy, originality, and integrity of the final manuscript, including all methods, results, and conclusions.

ETHICS STATEMENT

This work investigates computational methods for syndrome-extraction circuit design in quantum error correction, with performance assessed through numerical simulations. The study does not involve human participants, personal data, or sensitive information concerning individuals. Its intended contribution is to improve the reliability of quantum error-correction circuits. Within the scope of the methods and experiments presented, we do not identify specific ethical concerns or anticipate direct adverse impacts on individuals or society.

REPRODUCIBILITY STATEMENT

We are committed to supporting the independent verification and reproduction of the results presented in this work. During peer review, the source code and supporting experimental materials will be made available to reviewers upon reasonable request through channels that preserve the anonymity of the submission. Upon acceptance, we will publicly release the source code, benchmark specifications, experimental configurations, optimized circuits, and the result data and scripts needed to reproduce the reported tables and figures.

REFERENCES

Laleh Aghababaie Beni, Oscar Higgott, and Noah Shutty. Tesseract: A search-based decoder for quantum error correction, 2025. URL https://arxiv.org/abs/2503.10988.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024.

Benjamin Anker and Milad Marvian. Flag gadgets based on classical codes. *PRX Quantum*, 5(4): 040340, 2024.

Johannes Bausch, Andrew W Senior, Francisco JH Heras, Thomas Edlich, Alex Davies, Michael Newman, Cody Jones, Kevin Satzinger, Murphy Yuezhen Niu, Sam Blackwell, et al. Learning high-accuracy error decoding for quantum processors. *Nature*, 635(8040):834–840, 2024.

Michael E Beverland, Shilin Huang, and Vadym Kliuchnikov. Fault tolerance of stabilizer channels. *arXiv preprint arXiv:2401.12017*, 2024.

Aditya Bhardwaj, Muzhou Ma, Nadine Meister, Robbie King, Dolev Bluvstein, John Preskill, Madelyn Cain, Qian Xu, and Hsin-Yuan Huang. High-rate qldpc processors. *arXiv preprint arXiv:2607.28795*, 2026.

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.

A Robert Calderbank and Peter W Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098, 1996.

Earl T Campbell, Barbara M Terhal, and Christophe Vuillot. Roads towards fault-tolerant universal quantum computation. *Nature*, 549(7671):172–179, 2017.

Christopher Chamberland and Michael E Beverland. Flag fault-tolerant error correction with arbitrary distance codes. *Quantum*, 2:53, 2018.

Rui Chao and Ben W Reichardt. Quantum error correction with only two extra qubits. *Physical review letters*, 121(5):050502, 2018.

Ben Criger, Aaron Hankin, Justin J Burau, and Abigail R Perry. Automated flag-based fault-tolerant state preparation using integer linear programming. *arXiv preprint arXiv:2607.22498*, 2026.

Peter-Jan HS Derks, Alex Townsend-Teague, Ansgar G Burchards, and Jens Eisert. Designing fault-tolerant circuits using detector error models. *Quantum*, 9:1905, 2025.

Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search for variational quantum algorithms. *npj Quantum Information*, 8(1):62, 2022.

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`.

Craig Gidney, Michael Newman, Austin Fowler, and Michael Broughton. A fault-tolerant honeycomb memory. *Quantum*, 5:605, 2021. doi: 10.22331/q-2021-12-20-605. URL `https://quantum-journal.org/papers/q-2021-12-20-605/`.

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

Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. *arXiv preprint arXiv:0904.2557*, 2009.

Zhimin He, Hongxiang Chen, Yan Zhou, Haozhen Situ, Yongyao Li, and Lvzhou Li. Self-supervised representation learning for Bayesian quantum architecture search. *Physical Review A*, 111(3): 032403, 2025. doi: 10.1103/PhysRevA.111.032403.

Zhimin He, Zhengjie Zeng, Haozhen Situ, Shenggen Zheng, Yan Zhou, and Lvzhou Li. Realigning quantum architecture search with a top-tier-focused training paradigm for performance predictors. *Physical Review Applied*, 26(1):014094, 2026. doi: 10.1103/wz9c-h89g.

Shanchuan Li, Daisuke Tsukayama, Jun-ichi Shirakashi, Tetsuo Shibuya, and Hiroshi Imai. Quantum architecture search with neural predictor based on zx-calculus. *EPJ Quantum Technology*, 12 (1):106, 2025.

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

Pei-Hao Liou and Ching-Yi Lai. Parallel syndrome extraction with shared flag qubits for calderbank-shor-steane codes of distance three. *Physical Review A*, 107(2):022614, 2023.

Pei-Hao Liou and Ching-Yi Lai. Reducing quantum error correction overhead with versatile flag-sharing syndrome extraction circuits. *IEEE Transactions on Quantum Engineering*, 6:1–24, 2025.

Yuhao Liu, Shuohao Ping, Junyu Zhou, Ethan Decker, Justin Kalloor, Mathias Weiden, Kean Chen, Yunong Shi, Ali Javadi-Abhari, Costin Iancu, et al. Alphasyndrome: Tackling the syndrome measurement circuit scheduling problem for qec codes. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 77–93, 2026.

Xudong Lu, Kaisen Pan, Ge Yan, Jiaming Shan, Wenjie Wu, and Junchi Yan. QAS-bench: Rethinking quantum architecture search and a benchmark. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22880–22898. PMLR, 2023. URL https://proceedings.mlr.press/v202/lu23f.html.

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. Degenerate quantum ldpc codes with good finite length performance. *Quantum*, 5:585, 2021a.

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

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.

Balint Pato, Theerapat Tansuwannont, Shilin Huang, and Kenneth R Brown. Optimization tools for distance-preserving flag fault-tolerant error correction. *PRX Quantum*, 5(2):020336, 2024.

Tom Peham, Ludwig Schmid, Lucas Berent, Markus Müller, and Robert Wille. Automated synthesis of fault-tolerant state preparation circuits for quantum error-correction codes. *PRX quantum*, 6 (2):020330, 2025.

Laurent Perron, Frédéric Didier, and Steven Gay. The cp-sat-lp solver (invited talk). In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

Leonid P. Pryadko, Vadim A. Shabashov, and Valerii K. Kozin. QDistRnd: A GAP package for computing the distance of quantum error-correcting codes. *Journal of Open Source Software*, 7 (71):4120, 2022. doi: 10.21105/joss.04120. URL https://doi.org/10.21105/joss.04120.

Joschka Roffe. Quantum error correction: an introductory guide. *Contemporary Physics*, 60(3): 226–245, 2019.

Mackenzie H Shaw and Barbara M Terhal. Lowering connectivity requirements for bivariate bicycle codes using morphing circuits. *Physical review letters*, 134(9):090602, 2025.

Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):R2493–R2496, 1995. doi: 10.1103/PhysRevA.52.R2493.

Andrew Steane. Multiple-particle interference and quantum error correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 452(1954): 2551–2577, 1996a.

Andrew M. Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5): 793–797, 1996b. doi: 10.1103/PhysRevLett.77.793.

Armands Strikis, Dan E Browne, and Michael E Beverland. High-performance syndrome extraction circuits for quantum codes. *arXiv preprint arXiv:2603.05481*, 2026.

Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307–346, 2015.

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.

Maxime A Tremblay, Nicolas Delfosse, and Michael E Beverland. Constant-overhead quantum error correction with thin planar connectivity. *Physical Review Letters*, 129(5):050504, 2022.

Joshua Viszlai, Satvik Maurya, Swamit Tannu, Margaret Martonosi, and Frederic T Chong. Prophunt: Automated optimization of quantum syndrome measurement circuits. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1476–1491, 2026.

Suhas Vittal, Ali Javadi-Abhari, Andrew W Cross, Lev S Bishop, and Moinuddin Qureshi. Flagproxy networks: Overcoming the architectural, scheduling and decoding obstacles of quantum ldpc codes. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 718–734. IEEE, 2024.

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.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Wenjie Wu, Ge Yan, Xudong Lu, Kaisen Pan, and Junchi Yan. Quantumdarts: differentiable quantum architecture search for variational quantum algorithms. In *International conference on machine learning*, pp. 37745–37764. PMLR, 2023.

Qian Xu, J Pablo Bonilla Ataides, Christopher A Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasić, Mikhail D Lukin, Liang Jiang, and Hengyun Zhou. Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays. *Nature Physics*, 20(7):1084–1090, 2024.

Ge Yan, Shanchuan Li, and Yuxuan Du. Rethink the role of neural decoders in quantum error correction. *arXiv preprint arXiv:2605.12046*, 2026a.

Ge Yan, Shanchuan Li, Shiyi Xiao, Pengyue Ma, Hanyan Cao, Feng Pan, and Yuxuan Du. Efficient foundation decoders for fault-tolerant quantum computing. *arXiv preprint arXiv:2606.27119*, 2026b.

Ge Yan, Wenjie Wu, Yuheng Chen, Kaisen Pan, Xudong Lu, Zixiang Zhou, Yuhan Wang, Ruocheng Wang, and Junchi Yan. Ai-empowered quantum circuit synthesis and compilation optimization: Overview and prospects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026c.

John Zhuoyang Ye, Aarav Pabla, and Jens Palsberg. Reinforcement learning for syndrome extraction. *arXiv preprint arXiv:2609.12020*, 2026.

Keyi Yin, Hezi Zhang, Xiang Fang, Yunong Shi, Travis S Humble, Ang Li, and Yufei Ding. Qeccsynth: A layout synthesizer for quantum error correction codes on sparse architectures. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 876–890, 2025.

Theodore J Yoder and Isaac H Kim. The surface code with a twist. *Quantum*, 1:2, 2017.

Remmy Zen, Jan Olle, Luis Colmenarez, Matteo Puviani, Markus Müller, and Florian Marquardt. Quantum circuit discovery for fault-tolerant logical state preparation with reinforcement learning. *Physical Review X*, 15(4):041012, 2025.

Kai Zhang, Dingchao Gao, Zhaohui Yang, Runshi Zhou, Fangming Liu, Zhengfeng Ji, and Jianxin Chen. Optimal compilation of syndrome extraction circuits for general quantum ldpc codes. In *2026 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7. IEEE, 2026.

Shi-Xin Zhang, Chang-Yu Hsieh, Shengyu Zhang, and Hong Yao. Differentiable quantum architecture search. *Quantum Science & Technology*, 7(4):045023, 2022.

A EXTENDED PRELIMINARIES

A.1 STABILIZER AND CSS QUANTUM CODES

Stabilizer codes. Consider n physical qubits with Hilbert space $(\mathbb{C}^2)^{\otimes n}$, and let $\mathsf{Pauli}_n = \{\pm 1, \pm i\} \cdot \{I, X, Y, Z\}^{\otimes n}$ denote the n-qubit Pauli group. An $[\![n, k, d]\!]$ stabilizer code is specified by an Abelian subgroup $\mathcal{S} \subset \mathsf{Pauli}_n$ with $-I \notin \mathcal{S}$. Its codespace is

$$\left\{ |\psi\rangle \in (\mathbb{C}^2)^{\otimes n} : S\,|\psi\rangle = |\psi\rangle \text{ for all } S \in \mathcal{S} \right\}. \tag{11}$$

If $\mathcal{S}$ has $n - k$ independent generators, then the codespace has dimension 2^k.

The Pauli normalizer of $\mathcal{S}$ is $\mathrm{PauliNorm}(\mathcal{S}) = \{E \in \mathsf{Pauli}_n : ES = SE,\ \forall S \in \mathcal{S}\}$. Logical Pauli operators, identified up to global phase, are represented by $\mathrm{PauliNorm}(\mathcal{S})/\langle iI, \mathcal{S}\rangle$. The code distance is

$$d = \min_{E \in \mathrm{PauliNorm}(\mathcal{S}) \setminus \langle iI, \mathcal{S}\rangle} \mathrm{wt}(E), \tag{12}$$

where $\mathrm{wt}(E)$ counts its nonidentity tensor factors. Consequently, an arbitrary Pauli error of weight at most $\lfloor (d-1)/2 \rfloor$ is correctable.

Syndromes and logical classes. Fix a possibly redundant generating list $S_1, \ldots, S_m$ of $\mathcal{S}$. The syndrome of a Pauli error E is the binary vector $\sigma(E) \in \mathbb{F}_2^m$ defined by $S_i E = (-1)^{\sigma_i(E)} E S_i$. Two errors E and E' have the same syndrome precisely when $E^\dagger E' \in \mathrm{PauliNorm}(\mathcal{S})$. Within a fixed syndrome, they belong to the same logical class if they differ by a stabilizer up to global phase, and to different logical classes otherwise. Decoding therefore identifies a logical class compatible with the observed syndrome, rather than a unique physical error.

Calderbank-Shor-Steane (CSS) codes. A CSS code admits generating sets consisting separately of X-type and Z-type Pauli operators. Let $H_X \in \mathbb{F}_2^{m_X \times n}$ and $H_Z \in \mathbb{F}_2^{m_Z \times n}$ denote their binary support matrices. For a binary row vector $v \in \mathbb{F}_2^n$, write $X(v) = \prod_{j:v_j=1} X_j$ and $Z(v) = \prod_{j:v_j=1} Z_j$. The commutation condition and the number of logical qubits are

$$H_X H_Z^\top = 0, \qquad k = n - \operatorname{rank} H_X - \operatorname{rank} H_Z, \tag{13}$$

with all binary linear-algebra operations taken over $\mathbb{F}_2$. Up to global phase, a Pauli error can be written as $E = X(e_X)Z(e_Z)$, where $e_X, e_Z \in \mathbb{F}_2^n$ are row vectors. Its ideal syndromes are $s_X = H_X e_Z^\top$ and $s_Z = H_Z e_X^\top$. For these CSS checks, taking the generating list in the order of the checks specified by the rows of H_X, followed by those of H_Z, gives $m = m_X + m_Z$ and $\sigma(E) = (s_X^\top, s_Z^\top)^\top$.

The logical Pauli classes in the X and Z sectors form, respectively, the quotient spaces

$$\ker H_Z / \operatorname{row} H_X, \qquad \ker H_X / \operatorname{row} H_Z. \tag{14}$$

Here the kernels are identified with spaces of binary row vectors via transposition: $v \in \ker H_Z$ means $H_Z v^\top = 0$, and similarly for H_X. The sector distances d_X and d_Z are the minimum Hamming weights of representatives of nonzero classes in these two quotient spaces, respectively, with $d = \min(d_X, d_Z)$.

A.2 ABELIAN TWO-BLOCK GROUP-ALGEBRA CODES

Finite Abelian group representation. Let $G = C_{\ell_1} \times \cdots \times C_{\ell_t}$ be a finite Abelian group of order $N = \prod_{j=1}^t \ell_j$, where $C_{\ell_j} \cong \mathbb{Z}/\ell_j\mathbb{Z}$. We write elements of G as tuples $g = (g^{(1)}, \ldots, g^{(t)})$, with addition and negation performed componentwise modulo $(\ell_1, \ldots, \ell_t)$. Let $\mathcal{A}, \mathcal{B} \subseteq G$ be the supports of two elements of the binary group algebra $\mathbb{F}_2[G]$. For a subset $\mathcal{U} \subseteq G$, define its translation matrix $T(\mathcal{U}) \in \mathbb{F}_2^{N \times N}$ by

$$[T(\mathcal{U})]_{g,h} = \mathbf{1}[h - g \in \mathcal{U}], \qquad g, h \in G. \tag{15}$$

Equivalently, row g has nonzero entries precisely at the columns $h = g + u$ with $u \in \mathcal{U}$, where addition is performed in G.

Two-block CSS construction. Set $A = T(\mathcal{A})$ and $B = T(\mathcal{B})$. The corresponding Abelian two-block group-algebra code is specified by

$$H_X = [A \mid B], \qquad H_Z = [B^\top \mid A^\top]. \tag{16}$$

Both matrices have N rows and $2N$ columns, so $m_X = m_Z = N$ and the code acts on $n = 2N$ data qubits. Since translation matrices associated with an Abelian group commute, $AB = BA$, and hence

$$H_X H_Z^\top = AB + BA = 0 \tag{17}$$

over $\mathbb{F}_2$. The two matrices therefore define a valid CSS code with $k = 2N - \operatorname{rank} H_X - \operatorname{rank} H_Z$. Each measured check has weight $w = |\mathcal{A}| + |\mathcal{B}|$.

This construction includes cyclic generalized-bicycle codes (Lin & Pryadko, 2024) and the bivariate-bicycle (BB) codes of Bravyi et al. (2024). In the latter construction, $G = C_{\ell_1} \times C_{\ell_2}$, where ℓ_1 and ℓ_2 are the orders of the two cyclic factors, corresponding to the translation periods along the two coordinate directions. The original BB construction uses $|\mathcal{A}| = |\mathcal{B}| = 3$, yielding weight-six checks on $n = 2\ell_1\ell_2$ data qubits.

Group-indexed checks and data qubits. The data qubits are divided into two blocks, $\{L_g : g \in G\}$ and $\{R_g : g \in G\}$. The X- and Z-type checks are likewise indexed by G. For a support set $\mathcal{U} \subseteq G$, write $-\mathcal{U} = \{-u : u \in \mathcal{U}\}$. The translation matrices satisfy $T(\mathcal{U})^\top = T(-\mathcal{U})$, so transposition negates the displacements. For $g \in G$, the stabilizer operators are

$$S_g^X = \prod_{u\in\mathcal{A}} X_{L_{g+u}} \prod_{u\in\mathcal{B}} X_{R_{g+u}}, \tag{18}$$

$$S_g^Z = \prod_{u\in-\mathcal{B}} Z_{L_{g+u}} \prod_{u\in-\mathcal{A}} Z_{R_{g+u}}. \tag{19}$$

For the globally indexed check $i \in \{1, \ldots, m_X + m_Z\}$ in the generating list of Appendix A.1, let $\alpha \in \{X, Z\}$ be its Pauli sector and $g_i \in G$ its group index. Then $S_i = S_{g_i}^\alpha$. Thus, all checks within a Pauli sector share the same local incidence pattern and differ only by a group translation. The N checks in either sector need not be linearly independent; their dependencies impose deterministic parity relations on the ideal syndrome outcomes.

Subgroups and cosets. For a subgroup $H \leq G$, the group decomposes into disjoint cosets

$$G = \bigsqcup_{c=1}^{C} (g_c + H), \qquad C = [G : H] = \frac{|G|}{|H|}. \tag{20}$$

Here g_c is a chosen representative of the c-th coset. The two limiting cases are $H = G$, which yields a single coset, and $H = \{0\}$, which yields one coset per group element. Within a fixed Pauli sector, all checks have the same set of relative interaction labels and act on group-translated data qubits. The subgroup H specifies the sharing groups for local data-CNOT ordering templates: checks whose group indices lie in the same coset share one template, with separate templates for the X and Z sectors. There are C templates per sector, or $2C$ templates in total.

A.3 Syndrome extraction and hook errors

Ancilla-based stabilizer measurement. Consider the local measurement of a CSS check i in Pauli sector $\alpha \in \{X, Z\}$, with weight w_i, syndrome ancilla a_i, and local data-CNOT ordering π_i. Its stabilizer operator is $S_i = \prod_{j=1}^{w_i} \alpha_{\pi_i(j)}$, where $\pi_i(j)$ is the physical data qubit visited in the j-th data CNOT. Figure 4(a,b) illustrates the corresponding measurement circuits. Define the data-CNOT unitary

$$W_i(\pi_i) = \begin{cases} \mathrm{CX}(\pi_i(w_i) \to a_i) \cdots \mathrm{CX}(\pi_i(1) \to a_i), & \alpha = Z, \\ \mathrm{CX}(a_i \to \pi_i(w_i)) \cdots \mathrm{CX}(a_i \to \pi_i(1)), & \alpha = X, \end{cases} \tag{21}$$

Here, each arrow points from the control qubit to the target qubit, and the rightmost gate is applied first.

For a Z-type check, the syndrome ancilla a_i is prepared in $|0\rangle$, used as the target of the data CNOTs, and measured in the Z basis. For an X-type check, a_i is prepared in $|+\rangle$, used as the control, and measured in the X basis. Equivalently, the X-basis preparation and measurement are implemented

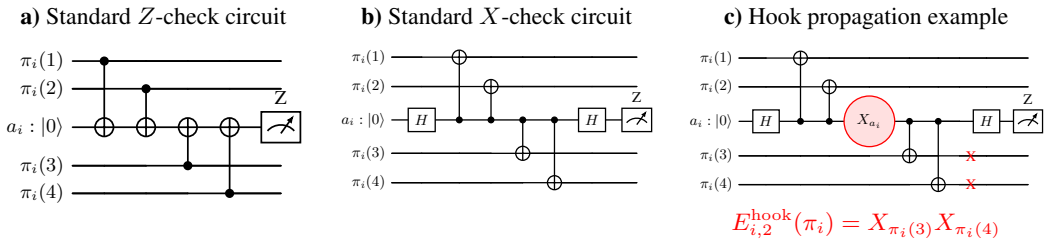

Figure 4: **Ancilla-based stabilizer measurements and hook-error propagation. a)** A weight-four Z-type check is measured using the data qubits as CNOT controls and the syndrome ancilla a_i as the target. **b)** A weight-four X-type check is measured using a_i as the CNOT control, with Hadamard gates implementing its preparation and readout in the X basis. **c)** An X_{a_i} fault occurring after the second data interaction propagates through the subsequent CNOTs and produces the correlated data residual $E^{\text{hook}}_{i,2}(\pi_i) = X_{\pi_i(3)}X_{\pi_i(4)}$. Red labels on the data wires indicate propagated Pauli components, not additional applied gates or independent faults.

by Hadamard gates before and after the CNOT sequence. The corresponding Heisenberg-picture identities are

$$\begin{aligned} W_i(\pi_i)^\dagger Z_{a_i} W_i(\pi_i) &= Z_{a_i} S_i, \qquad \alpha = Z, \\ W_i(\pi_i)^\dagger X_{a_i} W_i(\pi_i) &= X_{a_i} S_i, \qquad \alpha = X. \end{aligned} \tag{22}$$

Since the ancilla is initialized in the $+1$ eigenstate of its measurement observable, its final outcome reports the eigenvalue of the desired stabilizer. Equation (22) is independent of π_i; different orderings implement the same ideal measurement but generally exhibit different fault-propagation behaviour.

Pauli propagation through CNOT gates. Let W_{CX} denote a CNOT gate whose control and target qubit positions are labelled ctrl and tgt, respectively. For $\nu_1, \nu_2 \in \mathbb{F}_2$, write $|\nu_1, \nu_2\rangle = |\nu_1\rangle_{\text{ctrl}} \otimes |\nu_2\rangle_{\text{tgt}}$. Its action on the computational basis is

$$W_{\text{CX}} |\nu_1, \nu_2\rangle = |\nu_1, \nu_1 \oplus \nu_2\rangle . \tag{23}$$

A Pauli error E occurring immediately before the gate is propagated to $W_{\text{CX}} E W_{\text{CX}}^\dagger$ after the gate. Since $W_{\text{CX}}^\dagger = W_{\text{CX}}$, the propagation of an X error on the control follows directly from

$$\begin{aligned} W_{\text{CX}}(X_{\text{ctrl}} \otimes I_{\text{tgt}}) W_{\text{CX}}^\dagger |\nu_1, \nu_2\rangle &= W_{\text{CX}}(X_{\text{ctrl}} \otimes I_{\text{tgt}}) |\nu_1, \nu_1 \oplus \nu_2\rangle \\ &= W_{\text{CX}} |\nu_1 \oplus 1, \nu_1 \oplus \nu_2\rangle \\ &= |\nu_1 \oplus 1, \nu_2 \oplus 1\rangle \\ &= (X_{\text{ctrl}} \otimes X_{\text{tgt}}) |\nu_1, \nu_2\rangle . \end{aligned} \tag{24}$$

Thus, an X component on the control is copied to the target. Likewise, for a Z error on the target,

$$\begin{aligned} W_{\text{CX}}(I_{\text{ctrl}} \otimes Z_{\text{tgt}}) W_{\text{CX}}^\dagger |\nu_1, \nu_2\rangle &= (-1)^{\nu_1 \oplus \nu_2} |\nu_1, \nu_2\rangle \\ &= (Z_{\text{ctrl}} \otimes Z_{\text{tgt}}) |\nu_1, \nu_2\rangle , \end{aligned} \tag{25}$$

so a Z component on the target is copied to the control. The other two Pauli generators, Z_{ctrl} and X_{tgt}, remain unchanged. Collecting the four conjugation identities gives

$$\begin{aligned} W_{\text{CX}} X_{\text{ctrl}} W_{\text{CX}}^\dagger &= X_{\text{ctrl}} X_{\text{tgt}}, \\ W_{\text{CX}} Z_{\text{ctrl}} W_{\text{CX}}^\dagger &= Z_{\text{ctrl}}, \\ W_{\text{CX}} X_{\text{tgt}} W_{\text{CX}}^\dagger &= X_{\text{tgt}}, \\ W_{\text{CX}} Z_{\text{tgt}} W_{\text{CX}}^\dagger &= Z_{\text{ctrl}} Z_{\text{tgt}}. \end{aligned} \tag{26}$$

In an X-check measurement, the syndrome ancilla is the control of each data CNOT. Therefore, an X component on the syndrome ancilla is propagated to every subsequently coupled data target. In a Z-check measurement, the syndrome ancilla is the target, and a Z component on that ancilla is propagated to every subsequently coupled data control. These are precisely the ancilla components that generate correlated hook errors. Moreover, the propagating component commutes with the final ancilla measurement observable: X_{a_i} commutes with the X-basis readout of an X check, while Z_{a_i}

commutes with the Z-basis readout of a Z check. A propagating ancilla fault can therefore produce a correlated data error without necessarily flipping the syndrome bit of the check in which it occurs.

Hook errors. Suppose that the propagating component appears immediately after the j-th data interaction of check i. For an X-type check, an X_{a_i} component produces the data residual

$$E_{i,j}^{\mathrm{hook}}(\pi_i) = \prod_{j'=j+1}^{w_i} X_{\pi_i(j')}, \tag{27}$$

whereas for a Z-type check, a Z_{a_i} component produces

$$E_{i,j}^{\mathrm{hook}}(\pi_i) = \prod_{j'=j+1}^{w_i} Z_{\pi_i(j')}. \tag{28}$$

Thus, one physical fault can generate a correlated error on $w_i - j$ data qubits. Such a multi-qubit residual generated by the propagation of a single circuit fault is referred to as a *hook error*. Figure 4(c) illustrates an X_{a_i} fault occurring after the second interaction of a weight-four X check. The fault is copied to the remaining data targets, producing $E_{i,2}^{\mathrm{hook}}(\pi_i) = X_{\pi_i(3)}X_{\pi_i(4)}$.

The significance of a hook is that several data errors can arise from a single physical fault. If such correlated errors combine into an undetected nontrivial logical operation, fewer physical faults may suffice than the code distance would suggest. The relevant circuit-level quantity is d_{circ}, the minimum number of physical faults producing an undetected nontrivial logical effect for a fixed fault model and extraction protocol.

The presence of hooks does not, however, imply a reduction in d_{circ}. Suitable CNOT orderings preserve the distance of surface-code memory circuits (Yoder & Kim, 2017); the $[\![72, 12, 6]\!]$ BB memory circuit studied in (Aghababaie Beni et al., 2025) likewise retains $d_{\mathrm{circ}} = 6$. These examples show that the logical impact of a hook depends on both its propagated Pauli pattern and the information available to detect it. Flag measurements provide additional information about propagation faults and can help distinguish otherwise ambiguous error patterns.

A.4 Flagged stabilizer measurement

Flag circuit and ideal action. Flagged stabilizer measurements use an additional ancilla to obtain information about propagation faults (Chamberland & Beverland, 2018). Consider a check i with $M_i = 1$, syndrome ancilla a_i, and dedicated flag ancilla f_i. Here, f_i identifies the flag associated with an individual check. The labels f_X and f_Z in Section 2 indicate whether the associated check is X- or Z-type. For an X-type check, an X component on the syndrome ancilla a_i can propagate to the data without changing its X-basis readout. To detect this component, a flag ancilla f_i is initialized in $|0\rangle$ and measured in the Z basis, with two $\mathrm{CX}(a_i \to f_i)$ gates coupling the ancillas. For a Z-type check, the propagating component is a Z error on a_i; the corresponding flag f_i is initialized in $|+\rangle$, measured in the X basis, and coupled through two $\mathrm{CX}(f_i \to a_i)$ gates.

The interval between the two syndrome–flag CNOTs forms the *flag window*. We use the maximal window specified in Section 3.2. The first flag CNOT follows the first data CNOT, and the second precedes the last data CNOT. Figure 5(a) illustrates this arrangement for a weight-four X-type check with local ordering π_i. In the absence of faults, the flag CNOTs commute with the intervening data CNOTs and cancel each other. The original stabilizer measurement is therefore preserved, and the flag outcome is deterministically 0. Any odd number of X_{a_i} errors within the flag window triggers a flag measurement outcome of 1.

Detection of hook faults. Figure 5(b) shows an X fault on a_i after the second data interaction. This fault propagates through the remaining data CNOTs, producing the hook $E_{i,2}^{\mathrm{hook}}(\pi_i) = X_{\pi_i(3)}X_{\pi_i(4)}$. Since the fault occurs inside the flag window, the closing flag CNOT also transfers an X component to f_i, flipping its Z-basis outcome to 1. Thus, the propagating fault leaves the syndrome ancilla's own readout unchanged but triggers the flag. For a Z-type check, a Z fault on a_i inside the window similarly propagates to f_i and flips its X-basis outcome.

The flag outcome supplements the syndrome by constraining the compatible error patterns. Recovery therefore uses the syndrome and flag record jointly, since errors with the same stabilizer syndrome may be distinguished by their flag outcomes and assigned different corrections.

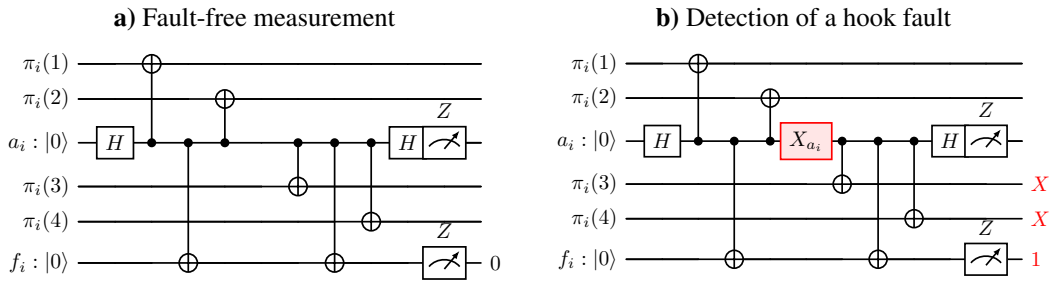

Figure 5: **Flagged measurement of a weight-four X check.** The data interactions visit $\pi_i(1), \ldots, \pi_i(4)$ in that order, and two additional CNOTs couple the syndrome ancilla a_i to the flag ancilla f_i after the first and before the last data interaction. **a)** Without faults, the circuit measures $S_i = \prod_{j=1}^{4} X_{\pi_i(j)}$ and the flag outcome is 0. **b)** A single X_{a_i} fault after the second data interaction produces the residual $E_{i,2}^{\text{hook}}(\pi_i) = X_{\pi_i(3)} X_{\pi_i(4)}$ and flips the flag outcome to 1. Red X labels at the data outputs denote propagated errors. All operations other than the indicated fault are ideal.

B Extended Related Work

Algebraic quantum codes. Algebraic constructions of quantum LDPC codes provide compact descriptions of sparse stabilizer checks, with the underlying structure influencing both code parameters and measurement circuits. Generalized bicycle codes use pairs of circulant matrices, while two-block group-algebra codes extend this construction to more general finite groups (Panteleev & Kalachev, 2021a; Lin & Pryadko, 2024). Bivariate and multivariate bicycle codes use commuting shifts to obtain families with different rates, distances, and connectivity (Bravyi et al., 2024; Voss et al., 2025). Such regularity also supports repeated measurement patterns, as illustrated by the translation-shared schedule in the BB memory construction (Bravyi et al., 2024). Alternative implementations, including morphing circuits, modify the measurement protocol to reduce connectivity requirements (Shaw & Terhal, 2025). Under circuit-level noise, the performance of these codes also depends on how faults propagate through extraction circuits. Stabilizer-channel and detector-error-model frameworks describe the resulting measurement records and logical effects, providing tools to characterize fault tolerance at the circuit level (Beverland et al., 2024; Derks et al., 2025).

Syndrome-extraction scheduling. Syndrome-extraction scheduling organizes data–ancilla interactions subject to qubit conflicts and measurement-compatibility constraints. Graph-based constructions use edge colouring to obtain parallel interaction layers (Tremblay et al., 2022). Constraint-based synthesis accommodates more flexible implementation choices: QECC-Synth combines ancilla bridges with satisfiability-based optimization for sparse hardware layouts, while ASC searches for low-depth schedules with compatible X/Z interactions (Yin et al., 2025; Zhang et al., 2026). Circuit reliability introduces an additional optimization criterion, since schedules with similar depths can propagate faults differently. ASC includes logical-error-rate filtering of generated candidates (Zhang et al., 2026). More directly, simulation-guided methods use decoding outcomes to steer scheduling decisions, through Monte Carlo tree search in AlphaSyndrome and reinforcement learning with importance sampling in FastSched (Liu et al., 2026; Ye et al., 2026). Fault-propagation-based methods offer a complementary approach. PropHunt combines local ambiguity analysis with MaxSAT-guided circuit modification, while residual-distance analyses connect propagated ancilla errors to circuit quality and support the construction of low-depth, nonuniform schedules (Viszlai et al., 2026; Strikis et al., 2026). These approaches differ in their search spaces and evaluation costs, spanning combinatorial depth optimization, structural fault analysis, and decoder-based performance optimization.

Flag-assisted fault tolerance. Flag circuits supplement stabilizer outcomes with information about faults that can propagate into correlated data errors. Foundational constructions show how this information supports fault-tolerant correction with few ancillary qubits and extends to codes of arbitrary distance under suitable correction conditions (Chao & Reichardt, 2018; Chamberland & Beverland, 2018). Classical error-correcting codes provide a systematic basis for designing flag

gadgets, including gadgets for multiple stabilizer measurements and settings without fast measurement and reset (Anker & Marvian, 2024). Subsequent work addresses the space and time overhead of these protocols through several complementary directions. Parallel extraction and physical flag sharing reduce ancilla requirements while coordinating CNOT orderings and syndrome information across rounds (Liou & Lai, 2023; 2025). Fault-set analysis, compact lookup tables, and adaptive temporal decoding further improve distance-preserving correction (Pato et al., 2024). Hardware-oriented designs such as Flag-Proxy Networks jointly consider connectivity, scheduling, and decoding with flag information (Vittal et al., 2024). Across these settings, resource efficiency depends on the interaction between additional fault information, ancillary operations, measurement depth, and idle noise.

Quantum architecture search and automated design. Quantum architecture search automates the selection of circuit structures using task-specific objectives (He et al., 2026; 2025; Li et al., 2025). For variational algorithms, parameter sharing reduces the cost of evaluating alternative architectures, while differentiable formulations support gradient-based structural optimization (Du et al., 2022; Wu et al., 2023; Yan et al., 2026c). Differentiable quantum architecture search (DQAS) optimizes a parameterized distribution over discrete circuits through score-function gradients, whereas QuantumDARTS uses differentiable relaxations for macro- and micro-level architecture choices (Zhang et al., 2022; Wu et al., 2023). Learning-based design also extends to error correction, including noise-adapted code optimization and joint discovery of codes and encoders (Nautrup et al., 2019; Olle et al., 2024). For fault-tolerant logical state preparation, reinforcement learning can construct preparation and flag-verification circuits together (Zen et al., 2025). Satisfiability-based methods synthesize fault-tolerant preparation circuits, while integer linear programming can select flag-based verification for a given preparation circuit (Peham et al., 2025; Criger et al., 2026). These lines of work consider different levels of the design problem, from code structure and encoding to auxiliary measurements and gate sequences, using either task evaluations or formal constraints to guide the search.

C Syndrome-extraction architecture details

We first describe circuit compilation for the general CSS architecture in Definition 2. We then explain how the group structure of Abelian 2BGA codes provides a template-sharing parameterization of this architecture.

C.1 Compilation constraints for general CSS codes

Given CSS check matrices H_X, H_Z satisfying $H_X H_Z^{\mathsf{T}} = 0$, the compiler constructs a global schedule τ for specified local orderings π and flag mask $\mathbf{M}$. Checks may have different weights w_i. As in Section 3.1, $\tau_i(j)$ denotes the positive integer tick assigned to the data CNOT between syndrome ancilla a_i and data qubit $\pi_i(j)$. The CNOT direction is $a_i \to \pi_i(j)$ for an X check and $\pi_i(j) \to a_i$ for a Z check.

Local ordering and flag placement. Each π_i lists every data qubit in the support of S_i exactly once. The compiler preserves this ordering through

$$\tau_i(j) < \tau_i(j+1), \qquad j = 1, \ldots, w_i - 1. \tag{29}$$

For a flagged check, let τ_i^{open} and τ_i^{close} denote the ticks of its two syndrome–flag CNOTs. The maximal-window gadget described in Appendix A.4 requires

$$\tau_i(1) < \tau_i^{\mathrm{open}} < \tau_i^{\mathrm{close}} < \tau_i(w_i), \tag{30}$$

together with

$$\tau_i^{\mathrm{open}} < \tau_i(j) < \tau_i^{\mathrm{close}}, \qquad j = 2, \ldots, w_i - 1. \tag{31}$$

Thus, the flag CNOTs occur after the first and before the last data CNOT and enclose all interior data interactions. We use this flag gadget for checks with $w_i \geq 3$. Checks of weight at most two are measured without a flag, with $M_i = 0$.

Qubit exclusivity. A qubit participates in at most one CNOT at each tick. Each syndrome ancilla, and each flag ancilla when present, is dedicated to one check. The local ordering and flag-window

constraints therefore exclude gate conflicts on these ancillas. For distinct checks sharing a data qubit, we impose

$$\tau_i(j) \neq \tau_{i'}(j') \quad \text{whenever} \quad \pi_i(j) = \pi_{i'}(j'), \qquad i \neq i'. \tag{32}$$

These constraints apply jointly to the X and Z checks.

Legality of interleaved measurements. The CSS condition ensures that every X–Z check pair has an even number of shared data qubits. A valid interleaved schedule must additionally satisfy the even-inversion condition of Zhang et al. (2026). For an X check S_i and a Z check $S_{i'}$, this is

$$\sum_{q \in \mathrm{supp}(S_i) \cap \mathrm{supp}(S_{i'})} \mathbf{1}\big[\tau_i\big(\pi_i^{-1}(q)\big) < \tau_{i'}\big(\pi_{i'}^{-1}(q)\big)\big] \equiv 0 \pmod 2, \tag{33}$$

where $\pi_i^{-1}(q)$ is the position of data qubit q in the local ordering π_i. The indicator therefore records whether the X-check interaction precedes the Z-check interaction at q.

Exchanging these two interactions introduces an additional CNOT from the X syndrome ancilla to the Z syndrome ancilla. An even number of such exchanges cancels this additional action, preserving the intended joint stabilizer measurement. For the dedicated flag gadgets used here, the two flag CNOTs commute with the intervening data interactions and cancel in the ideal circuit, as described in Appendix A.4. The same even-inversion condition therefore applies to the flagged architecture. In CP-SAT, each indicator is represented by a Boolean variable, and their sum is constrained to be twice an integer.

Compilation objective. Subject to these constraints, the compiler minimizes the final CNOT tick,

$$\min_{\tau} \max_{i} \tau_i(w_i). \tag{34}$$

Every flag CNOT precedes the final data CNOT of its check, so this objective includes both data and flag interactions. The constraints permit X/Z interleaving and depend only on the check supports, local orderings, and flag deployment. They therefore define the compilation problem for general CSS codes without requiring a group-based representation.

C.2 Template sharing and applicability to other CSS codes

From shared templates to check-specific orderings. For an Abelian 2BGA code, checks within each Pauli sector are group translations of a common interaction pattern. This provides a correspondence between their local data interactions, allowing multiple checks to share an ordering template. Following Section 3.1, a subgroup $H \leq G$ partitions the checks into $C = [G : H]$ cosets per sector, with one template $\bar{\pi}_c^{\alpha}$ for each coset c and sector $\alpha \in \{X, Z\}$.

For check i in Pauli sector α, with group index g_i in coset c, the template is expanded into its physical data-qubit ordering through

$$\pi_i(j) = \begin{cases} L_{g_i+u}, & \bar{\pi}_c^{\alpha}(j) = (L, u), \\ R_{g_i+u}, & \bar{\pi}_c^{\alpha}(j) = (R, u), \end{cases} \tag{35}$$

where addition is performed in G. The block label distinguishes interactions with the same displacement in the two data blocks. Thus, $\bar{\pi}_c^{\alpha}$ specifies a sequence of relative interactions, whereas π_i specifies the corresponding sequence of physical data qubits.

The grouped architecture $\widetilde{\gamma} = (H, \bar{\pi}, \mathbf{M}, \tau)$ therefore determines a general SEA $\gamma = (\pi, \mathbf{M}, \tau)$ through Eq. 35. The expanded orderings and flag mask are passed to the compiler described in Appendix C.1, as illustrated in Fig. 2. Template sharing leaves flag deployment check-specific: checks sharing an ordering template may have different values of M_i. Their execution ticks are also assigned individually by the global compiler.

Effect of subgroup selection. Subgroup selection controls the number of independently specified local orderings. Writing $N = |G|$, a fixed subgroup with index C requires $2C$ templates for the $2N$ measured checks. For check weight w, this gives $(w!)^{2C}$ possible local ordering assignments, compared with $(w!)^{2N}$ when every check has an independent ordering.

The choices $H = G$ and $H = \{0\}$ yield 2 and $2N$ templates, respectively. With $H = \{0\}$ included among the candidate subgroups, the parameterization can represent every collection of check-specific local orderings.

Application to other CSS codes. The general SEA representation also applies to hypergraph-product codes (Tillich & Zémor, 2014) and lifted-product codes (Panteleev & Kalachev, 2021b). For a given CSS code, each check can be assigned its own ordering distribution over its actual support. Checks with $w_i \geq 3$ additionally have a Bernoulli variable for flag deployment, while $M_i = 0$ for checks of weight at most two. The ordering distribution uses the check's weight w_i, so different checks need not have equal weights.

The sampled orderings and flag mask are compiled using the constraints above. If the code provides a correspondence between local interaction patterns, template sharing can additionally be introduced among compatible checks. Abelian 2BGA codes provide such a correspondence through group translation, giving the subgroup-based parameterization studied in this work.

C.3 Solver implementation and depth reporting

Reference schedules. For each sampled design, we first construct two feasible reference schedules. The first executes checks sequentially, preserving the complete local data and flag CNOT sequence of each check. The second schedules the X checks followed by the Z checks, greedily placing nonconflicting interactions in parallel within each sector. Both schedules are then compacted by moving gates to their earliest available ticks while preserving the gate order on every qubit. This compaction preserves the local ordering and X/Z legality constraints.

We select the shallower compacted schedule as the reference. Its depth provides an upper bound for the scheduling problem, and its gate ticks provide an initial solution hint to CP-SAT. The solver remains free to change the relative ordering of interactions from different checks and to interleave the X and Z sectors, subject to the constraints in Appendix C.1.

Solver configuration and termination. We use OR-Tools CP-SAT (Perron et al., 2023) to minimize the objective in Eq. 34. Each solver call uses one search worker and a time limit of 60 seconds. Different sampled designs are compiled independently and can be processed in parallel.

We retain the best feasible schedule found within the time limit. If the solver returns no feasible solution, we use the validated reference schedule. Empty ticks are removed from the returned schedule, and the resulting gate assignments are checked against the compilation constraints. The ideal stabilizer and flag measurement action is also verified before noisy circuit evaluation.

Depth reporting. The reported circuit depth is the number of nonempty CNOT layers in one syndrome-extraction round, including both data and flag CNOTs. Ancilla reset, single-qubit gates, and measurement layers are excluded from this depth count. These operations remain present, with their associated noise, in the memory circuits used to estimate LER.

D Search algorithm and optimization details

D.1 Probability parameterization and initialization

We implement the architecture distribution in equation 8 using trainable logits. Each candidate subgroup $H \leq G$ has one logit, and a softmax over all candidate subgroups defines $q_\theta(H)$. The candidate set contains all subgroups of the input group G.

For each candidate subgroup, we maintain a separate $w \times w$ table of logits for each coset and Pauli sector. Rows correspond to ordering positions j, and columns correspond to the relative interaction labels (β, u). At position j, a softmax over the remaining labels in row j determines the next interaction. The selected label is then removed from subsequent choices, producing the template $\bar{\pi}_c^\alpha$. Conditional on H, templates are sampled independently across cosets and Pauli sectors. Their parameters are specific to the selected subgroup.

Table 2: Optimization settings used for each benchmark code.

Setting	Value
Optimizer	Adam
Learning rate	0.05
Optimization updates	40
Candidates per update, K	128
Cost rescaling	Divide cost differences by 10^{-3}

Each check has one flag logit, whose sigmoid gives $q_\theta(M_i = 1)$. The mask entries are sampled independently, with separate parameters for individual X and Z checks.

The initial subgroup distribution assigns equal total probability to each attainable group count $C = [G : H]$, distributed uniformly among subgroups with that count. Ordering logits are initialized to zero, giving uniform sampling over complete permutations. Each flag-deployment probability is initialized to $1/2$.

D.2 Optimization settings and candidate selection

We optimize a separate set of probability parameters for each benchmark code. Table 2 lists the optimization settings, and Algorithm 1 summarizes the search procedure. All K candidates in each batch are sampled from the current distribution and contribute to the update. Parameters remain fixed during batch evaluation and are updated after the full batch has been evaluated, starting with the first batch. Circuit costs and baselines are held constant during differentiation; gradients propagate through the retained log-probabilities.

Algorithm 1: Batched probabilistic architecture search

Input: Input code and optimization settings
Output: Trained parameters θ and evaluated candidate records

Initialize θ as described above
Initialize Adam and an empty search history
for each optimization update **do**
    Draw $\xi_1, \ldots, \xi_K$ independently from q_θ
    Retain $\log q_\theta(\xi_\kappa)$ for every candidate
    for $\kappa = 1, \ldots, K$ **do**
        Compile $\tau_\kappa = \mathrm{Compile}(\xi_\kappa)$
        Evaluate $\widetilde{\gamma}_\kappa$ to obtain the LER estimate $\widehat{V}_\kappa$
        Record the circuit and its evaluation in the search history
    end for
    Compute the leave-one-out baselines $\overline{V}_{-\kappa}$
    Form the gradient estimate in equation 10, using the cost rescaling in Table 2
    Update θ using Adam
end for

For each code and experimental condition, we select the circuit with the lowest recorded LER estimate across the search history. The selected circuit is then held fixed and evaluated using fresh Monte Carlo samples independent of those used during search. The reported LER is obtained from this independent evaluation, following the protocol in Appendix E.2. The circuit distance, CNOT depth, flag use, and subgroup assignment all refer to this same circuit.

E Circuit-level evaluation

This appendix details the distance calculations and LER evaluation used for the benchmark circuits.

E.1 DISTANCE CALCULATION

Code-level distance. For Abelian 2BGA codes, group inversion combined with exchange of the two data blocks gives a weight-preserving correspondence between the two logical sectors (Lin & Pryadko, 2024). We verify this correspondence on the benchmark matrices and compute d in the X sector. The search therefore seeks the lowest-weight vector in $\ker H_Z$ that anticommutes with at least one operator in a complete Z-type logical basis.

Group translations reduce equivalent cases in the certification searches. Every nonempty logical support has a translated representative containing the group-identity qubit in one of the two data blocks. Restricting the search to these representatives preserves the minimum logical weight.

Circuit-level distance. For each selected circuit, we evaluate both memory bases using two syndrome-extraction rounds with noisy data preparation and terminal readout. Fault locations follow the SI1000 channels used for LER evaluation. The two bases are treated separately because the chosen orderings, flag deployment, and schedule need not preserve the code-level symmetry.

Each elementary fault has unit cost. Gate and idle locations include all nonidentity Pauli faults on the affected qubits, while preparation and measurement locations include the prescribed flips. A two-qubit Pauli at one CNOT location counts as one fault.

For each memory experiment, we construct binary matrices D and L whose columns record the detector and logical responses of individual fault mechanisms. The detector matrix includes all syndrome and flag detectors in that experiment, and the logical-response matrix contains a complete logical-observable basis. Let $\mathbf{f}$ be the binary vector selecting fault mechanisms. We solve

$$\min_{\mathbf{f}} \|\mathbf{f}\|_0 \qquad \text{subject to} \qquad D\mathbf{f} = 0, \quad L\mathbf{f} \neq 0. \tag{36}$$

All matrix operations are over $\mathbb{F}_2$. The constraints identify faults that leave every detector unchanged while producing a nontrivial logical effect. The reported d_{circ} is the smaller optimum of the two memory experiments.

Multiple Pauli faults selected at the same gate or idle location can be replaced by their product, which is either one allowed Pauli fault or the identity. This preserves their combined response without increasing the fault count. The minimum therefore agrees with the restriction to at most one physical fault per location.

Numerical search and certification. We obtain distance upper bounds using a randomized information-set decoding (ISD) search adapted from Pryadko et al. (2022). For circuit distance, each trial applies Gaussian elimination with a randomized column order to construct a basis of $\ker D$. The resulting vectors represent fault configurations with no detector response. We retain candidates satisfying $L\mathbf{f} \neq 0$ and refine them by adding kernel-basis vectors. Each refinement step selects the addition that most reduces Hamming weight while preserving a nonzero logical response.

Every candidate is verified against $D\mathbf{f} = 0$ and $L\mathbf{f} \neq 0$. Its weight therefore provides an upper bound on the circuit distance for that memory experiment. We retain the smallest verified weight across all trials and report the smaller bound from the two memory bases.

For all methods, each selected circuit is evaluated in both memory bases. Each basis uses five independent ISD searches, each with 2,000 iterations. Each iteration refines up to four low-weight nontrivial candidates and the best candidate found so far, with at most eight refinement steps per candidate. There is no ISD wall-clock limit, and early termination requires an exact distance certificate. CP-SAT additionally minimizes fault weight under the same detector and logical constraints. It uses one worker and a 300-second budget per memory basis, including model construction. Verified feasible solutions can improve the upper bound, while the global objective bound provides a lower bound. Matching lower and upper bounds certify the distance.

For code-distance searches, the same ISD procedure is applied to $\ker H_Z$, retaining vectors that anticommute with at least one operator in the complete Z-type logical basis. Code-distance searches refine up to three candidates per iteration, with at most four refinement steps per candidate.

Meet-in-the-middle certification. For a fault model, D and L record the detector and logical responses of its elementary fault mechanisms. Let $\mathcal{I}$ index these mechanisms and let $\mathbf{1}_J$ denote

Algorithm 2: Meet-in-the-middle certification for a fault model

Input: Response matrices D, L; fault indices $\mathcal{I}$; anchor indices $\mathcal{I}_0$ satisfying the stated symmetry condition; integer threshold $\rho \geq 1$; search budget
Output: An undetected logical fault of weight at most ρ, a certified lower bound $\rho+1$, or an incomplete search

$\rho_1 \leftarrow \lfloor(\rho-1)/2\rfloor$, $\rho_2 \leftarrow \lceil(\rho-1)/2\rceil$
Initialize an empty lookup table $\mathcal{H}$
for all $J_1 \subseteq \mathcal{I}$ with $|J_1| \leq \rho_1$ **do**
    if the search budget is exhausted **then**
        return incomplete
    end if
    $s \leftarrow D\mathbf{1}_{J_1}$, $\eta \leftarrow L\mathbf{1}_{J_1}$
    if $|\mathcal{H}[s]| < 2$ and η is not represented in $\mathcal{H}[s]$ **then**
        Add (η, J_1) to $\mathcal{H}[s]$
    end if
end for
for all $q \in \mathcal{I}_0$ **do**
    for all $J_2 \subseteq \mathcal{I}$ with $|J_2| \leq \rho_2$ **do**
        if the search budget is exhausted **then**
            return incomplete
        end if
        $s \leftarrow D(\mathbf{1}_{J_2} + \mathbf{1}_{\{q\}})$
        $\eta \leftarrow L(\mathbf{1}_{J_2} + \mathbf{1}_{\{q\}})$
        for all $(\eta', J_1) \in \mathcal{H}[s]$ **do**
            if $\eta' \neq \eta$ **then**
                $J \leftarrow J_1 \triangle J_2 \triangle \{q\}$
                return J
            end if
        end for
    end for
end for
return the certified lower bound $\rho + 1$

the binary indicator vector of a support $J \subseteq \mathcal{I}$. The corresponding responses are $D\mathbf{1}_J$ and $L\mathbf{1}_J$. We choose anchor indices $\mathcal{I}_0 \subseteq \mathcal{I}$ such that every nonempty support has a symmetry-equivalent representative containing an anchor. These symmetries preserve weight, zero detector response, and logical nontriviality.

Algorithm 2 searches for an undetected logical fault of weight at most ρ. It stores partial supports by detector response, retaining at most two distinct logical responses for each entry. A second enumeration combines another partial support with an anchor and searches for a matching detector response with a different logical response.

Here, $\triangle$ denotes symmetric difference. A returned support has zero detector response, nonzero logical response, and weight at most $\rho_1+\rho_2+1 = \rho$. Conversely, any undetected logical fault of weight at most ρ can be mapped by the stated symmetry to contain an anchor, then split within the two enumeration limits. For each detector response, a query excludes only one logical-response value, so retaining two distinct values preserves the existence of a match. Completing both enumerations without a match therefore certifies a distance of at least $\rho + 1$.

E.2 Circuit-level logical error rate

Memory experiments. We evaluate circuit performance using memory-X and memory-Z experiments with two syndrome-extraction rounds. For memory-Z, the data qubits are prepared in $|0\rangle^{\otimes n}$ and measured in the Z basis. For memory-X, preparation and readout use the X basis, implemented by adding Hadamard gates after reset and before measurement, respectively. Each extraction round measures both check sectors and includes the selected flag operations. Data preparation, basis changes, and terminal readout are included in the noise model. Terminal data measurements are aligned with the final-round ancilla measurements.

Individual detectors at the initial boundary use checks in the prepared memory basis. Subsequent rounds compare consecutive syndrome outcomes from both sectors, with deterministic stabilizer-

Table 3: SI1000 noise configuration. Depolarizing probabilities denote the total probability of a uniformly sampled nonidentity Pauli.

Operation	Noise channel and location	Probability
Single-qubit gate	Single-qubit depolarization after the gate	$p/10$
CNOT	Two-qubit depolarization after the gate	p
Reset	X error after preparation of $\|0\rangle$	$2p$
Measurement	X error before Z-basis measurement	$5p$
Idle	Single-qubit depolarization on qubits inactive during the time step	$p/10$
Measurement/reset waiting	Additional single-qubit depolarization on qubits not measured or reset during a time step containing measurement or reset	$2p$

Ordinary idle and measurement/reset waiting noise are applied independently when both conditions hold.

dependency and flag parities also included. Terminal check values are inferred from the data readout and combined with the final syndrome outcomes to form boundary detectors.

Noise model. We use the SI1000 noise model (Gidney et al., 2021) at $p = 0.003$, with the channels listed in Table 3.

Detector error model and decoding. We construct the detector error model (DEM) using Stim (Gidney, 2021), retaining correlated error mechanisms without graphlike decomposition or approximation of disjoint error channels.

We use the complete Telescoping decoder (S1–S4) (Bhardwaj et al., 2026). Its stages comprise GPU belief propagation (BP), GPU Relay-BP, CPU serial BP and Relay-BP, and a final integer-programming stage. The stage-specific DEM representations, acceptance rules, and parameter settings follow the reference memory-decoding configuration, with two reduced budgets. In S3-B, each variant uses 20 random relay legs instead of 40, in addition to its initial ordered leg. In S3-C, each configuration with random prior perturbations uses at most three seeds instead of ten; the configuration without random perturbations is run once. All compared methods use this common decoder configuration.

A block failure occurs when the decoder predicts at least one logical observable incorrectly or returns no accepted result after the final stage. For final evaluation, the selected circuit from each method is held fixed and evaluated using fresh Monte Carlo samples independent of those used during search or candidate selection. Sampling in each memory basis stops at 30 decoding failures or 10^6 shots, whichever occurs first. Sampling and decoding are batched, and only the sample prefix up to this stopping point contributes to the estimate. The block failure probability is estimated by the fraction of decoding failures among the retained samples.

Logical error per round and per logical qubit. For a code encoding k logical qubits, let $P^b_{\mathrm{block}}(R)$ denote the block failure probability after R extraction rounds in memory basis $b \in \{X, Z\}$. We use $R = 2$ in our evaluations. Following the reporting conventions of Wang et al. (2026); Bausch et al. (2024), we first define the effective failure probability per logical qubit as

$$P^b_{\mathrm{LQ}}(R) = 1 - \left(1 - P^b_{\mathrm{block}}(R)\right)^{1/k}. \tag{37}$$

This matches the block success probability to a reference model of k independent logical qubits with equal failure probabilities. Matching the round dependence to independent logical flips then gives

$$P^b_{\mathrm{L}} = \frac{1}{2}\left[1 - \left(1 - 2P^b_{\mathrm{LQ}}(R)\right)^{1/R}\right], \qquad P^b_{\mathrm{LQ}}(R) \le \frac{1}{2}. \tag{38}$$

We apply these two steps separately to the estimates from the two memory bases. The reported LER is the equally weighted average of P^X_{L} and P^Z_{L}.

F Experimental details

This appendix provides the benchmark constructions and baseline implementation details.

F.1 BENCHMARK CODE DETAILS

Table 4 specifies the 20 Abelian 2BGA CSS codes used in our experiments: five SB codes, four IBM BB codes, seven general BB codes, and four TB codes. The entries follow the order of the main results table.

The two polynomial columns specify the supports $\mathcal{A}$ and $\mathcal{B}$ through the exponent vectors of their nonzero monomials over $\mathbb{F}_2$. The variables x, y, z correspond to the cyclic factors of G in the listed order, with exponents reduced modulo the corresponding factor orders.

Table 4: Algebraic specifications and parameters of the 20 benchmark codes.

$[\![n, k, d]\!]$	G	w	Polynomial defining $\mathcal{A}$	Polynomial defining $\mathcal{B}$
			SB codes	
$[\![60, 4, 8]\!]$	C_{30}	5	$x + x^{10}$	$x^3 + x^{13} + x^{17}$
$[\![54, 4, 8]\!]$	C_{27}	6	$x^7 + x^{12} + x^{26}$	$x^6 + x^{25} + x^{26}$
$[\![60, 4, 9]\!]$	C_{30}	7	$x^5 + x^{10} + x^{27}$	$x^3 + x^4 + x^9 + x^{19}$
$[\![84, 6, 10]\!]$	C_{42}	7	$x^3 + x^{13} + x^{19}$	$x^7 + x^{27} + x^{31} + x^{36}$
$[\![42, 2, 7]\!]$	C_{21}	8	$x^3 + x^4 + x^{13} + x^{16}$	$x^7 + x^{15} + x^{18} + x^{19}$
			IBM BB codes	
$[\![72, 12, 6]\!]$	$C_6 \times C_6$	6	$x^3 + y + y^2$	$y^3 + x + x^2$
$[\![90, 8, 10]\!]$	$C_{15} \times C_3$	6	$x^9 + y + y^2$	$1 + x^2 + x^7$
$[\![108, 8, 10]\!]$	$C_9 \times C_6$	6	$x^3 + y + y^2$	$y^3 + x + x^2$
$[\![144, 12, 12]\!]$	$C_{12} \times C_6$	6	$x^3 + y + y^2$	$y^3 + x + x^2$
			General BB codes	
$[\![56, 6, 8]\!]$	$C_7 \times C_4$	6	$1 + x^2 + x^3y^2$	$1 + x^2y + x^3$
$[\![48, 4, 7]\!]$	$C_4 \times C_6$	7	$y^5 + x^2 + x^3y^4$	$y^4 + xy^2 + x^2y + x^2y^2$
$[\![72, 4, 8]\!]$	$C_6 \times C_6$	7	$xy^3 + x^3y^2 + x^4y^4$	$y + x + x^3y^4 + x^5y^3$
$[\![96, 8, 10]\!]$	$C_6 \times C_8$	7	$y^4 + xy^7 + x^2y^3$	$y^4 + x^2y + x^2y^4 + x^3y^2$
$[\![32, 2, 6]\!]$	$C_4 \times C_4$	8	$1 + y + y^3 + x^3y$	$xy^2 + x^2 + x^2y + x^3$
$[\![48, 2, 8]\!]$	$C_4 \times C_6$	8	$y + xy + x^2 + x^2y^3$	$1 + y^4 + xy^2 + x^2y^4$
$[\![64, 2, 10]\!]$	$C_4 \times C_8$	8	$1 + y + y^3 + x^3y$	$xy^2 + x^2 + x^2y + x^3$
			TB codes	
$[\![60, 4, 8]\!]$	$C_2 \times C_3 \times C_5$	7	$z^3 + xyz + xy^2z^4$	$z + y^2z^3 + xz^3 + xy^2z$
$[\![84, 6, 10]\!]$	$C_2 \times C_3 \times C_7$	6	$1 + y^2z^4 + xyz^5$	$1 + z + xyz^3$
$[\![90, 8, 9]\!]$	$C_3 \times C_3 \times C_5$	7	$1 + xy^2z^2 + x^2yz^3$	$xz + xy^2z + x^2z^4 + x^2y^2$
$[\![128, 20, 8]\!]$	$C_4 \times C_4 \times C_4$	8	$1 + x + y + z$	$1 + x^3 + y^3 + z^3$

F.2 BASELINE IMPLEMENTATIONS

All baseline evaluations use the common SI1000 noise model and Telescoping decoder configuration. Final LERs are obtained from independent evaluations of the fixed circuits selected by each baseline, following the protocol in Appendix E.2.

ASC. We use the authors' open-source implementation[1] of ASC (Zhang et al., 2026). The scheduling model assigns CNOT gates to layers subject to qubit-availability and X/Z compatibility constraints, allowing interactions from the two check sectors to interleave. For each solver run, we test increasing depth bounds using CP-SAT, with a time limit of 600 seconds per bound, and retain the first feasible schedule. The unflagged baseline uses $\mathbf{M} = \mathbf{0}$.

ASC+flag. We extend the ASC scheduling model with one dedicated flag ancilla per measured check, setting $\mathbf{M} = \mathbf{1}$. The two flag CNOTs are placed after the first and before the last data CNOT, respectively, implementing the same maximal flag window used in our method. Their timing

[1] https://github.com/iqubit-org/Auto-Stabilizer-Check

variables and ancilla-availability constraints are included in the scheduling model, so the flag and data CNOTs are scheduled jointly. The remaining compilation procedure is unchanged. For each benchmark code and each ASC variant, we generate ten candidate circuits and evaluate each using two extraction rounds. The candidate with the lowest estimated LER is selected and held fixed for the independent final evaluation.

AlphaSyndrome. We adapt the authors' open-source implementation[2] of AlphaSyndrome (Liu et al., 2026). We retain its unflagged MCTS scheduling procedure, which searches the X and Z sectors separately and concatenates their schedules with the X sector first. At each scheduling decision, the root is brought to 5,000 accumulated visits, including visits inherited through subtree reuse. The exploration coefficient is 1.4. The action with the highest empirical mean reward is committed, and its subtree is retained for the next decision. Each search is allowed up to 24 hours across both check sectors, including feedback evaluation. Search terminates when all scheduling decisions have been committed or when the time budget is exhausted.

Search feedback retains the original single-sector construction with one noisy extraction round bracketed by ideal Pauli-product measurements. An X-sector rollout uses Z-type stabilizer and logical parities, and a Z-sector rollout uses their X-type counterparts. Differences between the boundary measurements define the detectors and logical flips; syndrome-ancilla readout records are not used in these comparisons. We replace the original reward, which divides the sample count by one plus the failure count, with the sample count divided by the failure count. When no failure is observed, the reward equals the sample count.

On completion, the output uses the schedules obtained from the committed MCTS decisions. If the time budget is exhausted, the output uses the highest-reward fully evaluated rollout in the current sector, or the most recent fully evaluated rollout if none is available in that sector. An unsearched sector retains its greedy initialization. The fixed output circuit is then independently evaluated using fresh Monte Carlo samples, with both check sectors, two extraction rounds, and noisy preparation and readout. This final evaluation is excluded from the search time.

G Additional experimental results

G.1 Circuit depth and computational cost

To assess the depth tradeoff accompanying the LER improvements, Table 5 reports the CNOT depths of the same circuits evaluated in Table 1. Depth counts the nonempty CNOT layers in one syndrome-extraction round, including both data and flag CNOTs. Our selected circuits have depths between 9 and 16. Compared with ASC, they are deeper on 19 of the 20 benchmarks, while achieving lower LERs on all benchmarks in Table 1. Thus, smaller CNOT depth does not consistently correspond to better error-correction performance in these experiments. Compared with ASC+flag, our circuits are deeper on 15 codes, shallower on four, and equally deep on one. For example, on the $[\![128, 20, 8]\!]$ TB code, our circuit has depth 10, matching ASC and reducing the depth of ASC+flag from 12 to 10. Its LER is also lower than those of both baselines. These results show that the performance gains are accompanied by different depth tradeoffs across codes.

Compared with AlphaSyndrome, our circuits are shallower on all 20 benchmarks. The AlphaSyndrome circuits are two to six times as deep, corresponding to depth reductions of 50.0–83.3% with our method. Together with the lower LERs in Table 1, this comparison shows improvements in both circuit depth and error-correction performance.

Computational resources and timing. All methods were run with the same hardware allocation: one NVIDIA A100 GPU with 40 GB of memory and an AMD EPYC 7763 CPU. For our method and AlphaSyndrome, the reported wall-clock search time includes candidate generation, circuit compilation, and feedback evaluation. For ASC and ASC+flag, it includes only the generation of the candidate circuit; their LER evaluations are excluded.

[2] https://github.com/acasta-yhliu/asyndrome

Table 5: CNOT depth of the circuits reported in Table 1.

		CNOT depth			
$[\![n, k, d]\!]$	w	Ours	ASC	ASC+flag	AlphaSyndrome
Single-variable bicycle codes					
$[\![60, 4, 8]\!]$	5	10	5	7	20
$[\![54, 4, 8]\!]$	6	12	7	9	64
$[\![60, 4, 9]\!]$	7	13	8	10	69
$[\![84, 6, 10]\!]$	7	13	7	9	62
$[\![42, 2, 7]\!]$	8	10	9	11	60
IBM bivariate bicycle codes					
$[\![72, 12, 6]\!]$	6	10	7	9	32
$[\![90, 8, 10]\!]$	6	10	7	9	29
$[\![108, 8, 10]\!]$	6	9	8	10	38
$[\![144, 12, 12]\!]$	6	9	8	10	44
General bivariate bicycle codes					
$[\![56, 6, 8]\!]$	6	9	7	9	26
$[\![48, 4, 7]\!]$	7	11	7	9	33
$[\![72, 4, 8]\!]$	7	13	7	9	28
$[\![96, 8, 10]\!]$	7	15	7	9	36
$[\![32, 2, 6]\!]$	8	16	9	11	41
$[\![48, 2, 8]\!]$	8	14	8	10	41
$[\![64, 2, 10]\!]$	8	16	8	10	59
Trivariate bicycle codes					
$[\![60, 4, 8]\!]$	7	12	7	9	47
$[\![84, 6, 10]\!]$	6	12	8	9	41
$[\![90, 8, 9]\!]$	7	12	7	9	28
$[\![128, 20, 8]\!]$	8	10	10	12	59

G.2 Extended optimization

We examine longer training trajectories on four benchmark codes to assess the improvement available beyond the 40-update budget used in the main experiments. Figure 6 shows the learning dynamics over 100 optimization updates.

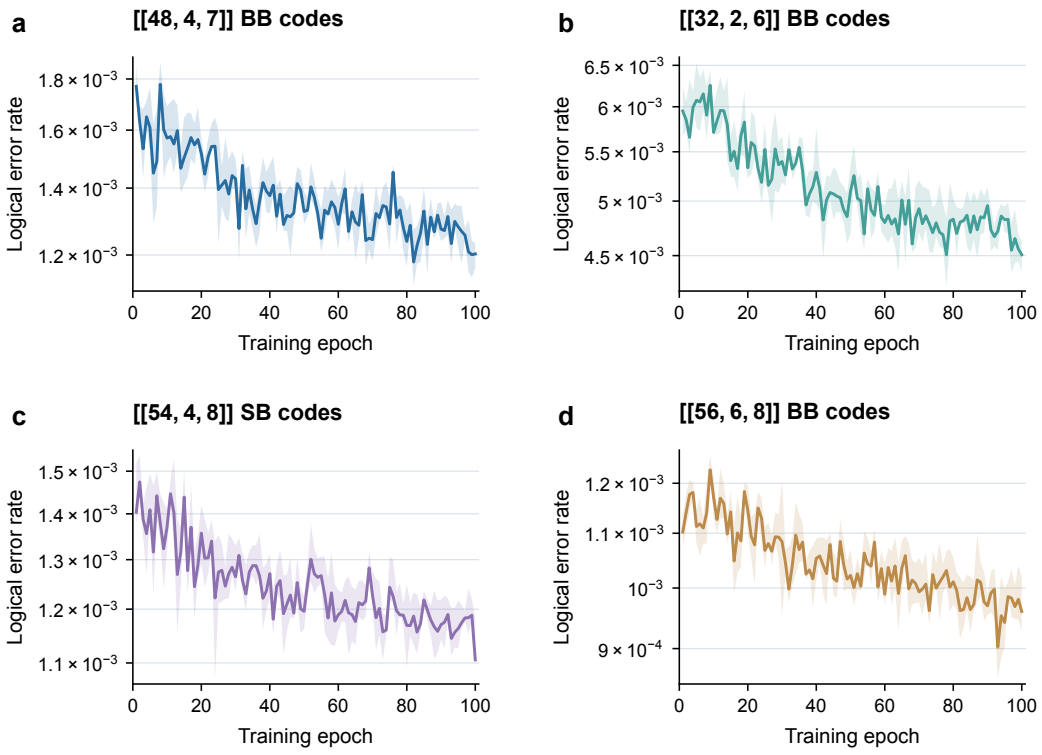

Figure 6: **Learning dynamics over 100 optimization updates for four benchmark codes.** Each curve shows the LER associated with the ten best candidates in each training batch. Shaded regions show the range from the lowest to the highest LER among these ten candidates.

All four codes exhibit further improvement beyond 40 updates, with visible fluctuations between batches. To quantify the additional gain, Table 6 compares the curve values averaged over updates 31–40 and 91–100. The reductions range from 7.60% to 11.18%, with the largest improvement observed for the $[\![32, 2, 6]\!]$ BB code.

Table 6: Improvement with extended optimization. Entries are averages of the curve values in Fig. 6 over the specified update intervals. LER values are in units of 10^{-3}.

Code	Updates 31–40	Updates 91–100	Relative reduction
$[\![48, 4, 7]\!]$ BB	1.3643	1.2527	8.18%
$[\![32, 2, 6]\!]$ BB	5.2786	4.6886	11.18%
$[\![54, 4, 8]\!]$ SB	1.2640	1.1662	7.74%
$[\![56, 6, 8]\!]$ BB	1.0458	0.9663	7.60%

The pattern of improvement differs between codes. For the $[\![48, 4, 7]\!]$ BB code, the average curve value decreases by 15.15% between updates 1–10 and 31–40, followed by a further 8.18% reduction between updates 31–40 and 91–100. For the $[\![32, 2, 6]\!]$ BB code, the corresponding reductions are 11.43% and 11.18%, showing substantial additional improvement in the longer training interval. These reductions indicate continued improvement among the ten best candidates sampled in each batch as the optimization budget increases.

G.3 Ablation settings and performance across physical error rates

Ablation settings. We evaluate three restricted variants on the five codes shown in Fig. 3(b). Table 7 summarizes the variables optimized in each variant. All variants use 40 optimization updates

Table 7: Search variables in the full method and its ablation variants.

Variant	Subgroup H	Ordering $\bar{\pi}$	Flag mask $\mathbf{M}$
Full method	Learned	Learned	Learned
Random π	Learned	Uniform random	Learned
No flags	Learned	Learned	$\mathbf{0}$
Full flags	Learned	Learned	$\mathbf{1}$

with 128 candidates per update, together with the same circuit compiler, noise model, and decoder as the full method.

In the Random π variant, each X- and Z-group template is sampled independently and uniformly from all permutations of its interaction labels. Fresh templates are drawn for every candidate, while checks within the same group share the sampled template. Subgroup selection and flag deployment remain learned from the complete-circuit feedback.

Performance across physical error rates. The preceding comparisons evaluate circuits at the noise level used during search. We further examine whether the resulting performance gains persist when the operating error rate changes. For each code and method, we hold the circuit selected at $p = 0.003$ fixed and evaluate it at $p = 0.001, 0.002, 0.003, 0.004$, and 0.005, using the same SI1000 noise model and decoder configuration. This evaluates the sensitivity of the selected architectures to noise strength.

Figure 7 shows that the full method achieves the lowest LER at every tested error rate on all five codes. Its advantage therefore extends to both lower and higher noise strengths than the search setting. The magnitude of this advantage nevertheless depends on the operating error rate. Relative to ASC+flag, the LER reductions range from 14.68–35.73% at $p = 0.003$ and 1.74–8.63% at $p = 0.005$. For the $[\![84, 6, 10]\!]$ TB code, for example, the reduction decreases from 35.73% to 5.20%. Thus, the selected circuits retain their performance advantage across the tested noise range, while the smaller margins at $p = 0.005$ show that the benefit of joint architecture optimization depends on noise strength.

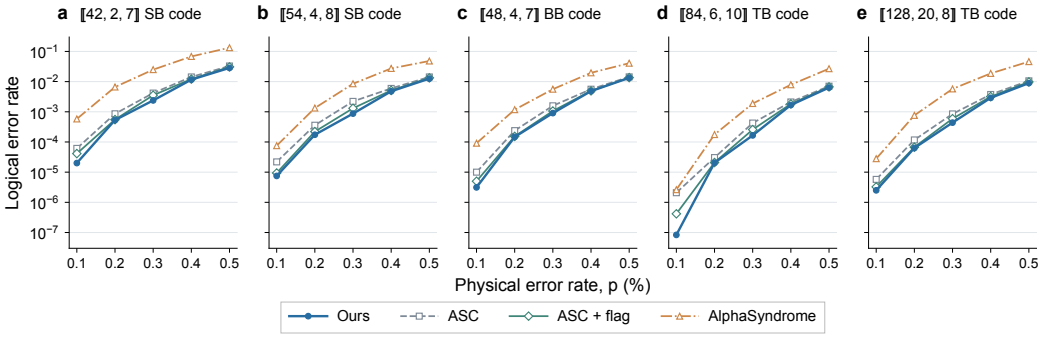

Figure 7: **Performance of fixed circuits across physical error rates.** Each panel compares the full method, ASC, ASC+flag, and AlphaSyndrome on one benchmark code. For each method, the circuit selected at $p = 0.003$ is evaluated across the noise range without changing its architecture. The LER axis is logarithmic.