COMMIT OR DEFER: A CHECKABLE NEURAL PRE-DECODER FOR QUANTUM LDPC CODES

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantum low-density parity-check (qLDPC) codes offer a route to lower qubit overhead in fault-tolerant quantum computing. However, algorithmic decoders can require costly iterative inference and candidate search, while standalone neural decoders often rely on large models and extensive training data. Accordingly, we propose CoD (Commit-or-Defer), a novel unified neural pre-decoding framework. Its lightweight predictor estimates fault posteriors by combining physical priors with gated recurrent unit (GRU) message passing on K-hop active subgraphs around fired detectors in a detector error model. A commit-or-defer strategy bypasses the classical backend when a candidate passes a zero-residual syndrome check and otherwise uses the same posteriors to guide backend inference. We bound the logical risk of committed shots in terms of posterior uncertainty and the DEM logical distance, and give a sufficient condition under which CoD lowers overall logical risk below the unassisted backend. A common circuit-level benchmark spanning four code families and three backends shows up to 42.8% lower mean logical error rates (LERs) and 55–93% fewer backend calls than unassisted decoding. On BB72 and BB144 under code-capacity noise, CoD achieves lower mean LERs than a standard graph neural decoder using roughly three million training shots and 42–$56\times$ fewer neural inference FLOPs.

1 INTRODUCTION

Noise limits the reliability of quantum computation [38], and quantum error correction (QEC) provides the foundation for protecting logical information [41, 43] as processors scale [21, 4, 22]. Surface codes benefit from mature, efficient decoders, but strong logical error suppression comes at a substantial cost in physical qubits, because each logical qubit occupies its own block whose size grows quadratically with the code distance [16, 25]. Higher-rate quantum low-density parity-check (qLDPC) families, exemplified by bivariate bicycle (BB) codes, encode many logical qubits in one block and offer a route to substantially lower overhead [8, 7]. This advantage, however, has to be paid for in decoding. The efficiency of surface-code decoders rests on a structure that qLDPC codes lack: every fault flips at most two detectors, so inference reduces to matching on a graph [14, 25]. In a qLDPC code each fault flips several high-weight checks, so the decoding graph becomes a hypergraph with short cycles and stabilizer-induced symmetries, and many distinct fault patterns reproduce the same syndrome while differing in their logical effect [37, 39, 29]; circuit-level noise adds correlated faults across syndrome-extraction rounds [7]. Decoding qLDPC codes is therefore a genuinely non-local inference problem, and the decoders that make these codes practical have to balance the accuracy of that inference against its cost on every shot [7, 34].

Existing decoders strike this balance in one of two ways. On the one hand, algorithmic decoders such as BP-OSD and Relay-BP rely on belief propagation (BP), which can fail to reproduce the syndrome on qLDPC codes even for sparse, local faults, as its marginals split between degenerate configurations [37, 29]. To compensate, BP-OSD solves for a consistent correction by Gaussian elimination over the reliability-ordered mechanisms, a step cubic in the code size, and Relay-BP re-runs BP with disordered memory strengths, each leg another pass over the whole decoding graph [36, 39, 34]. Because small, well-separated fault clusters are common at low physical error rates [26], the cost of handling these cases is central to overall decoding efficiency. Yet neither method uses information learned from data: both decode from the physical error rates alone, their only syndrome-conditioned input being BP's own marginals, so the local patterns that recur across shots are re-solved from

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.

scratch each time rather than recognised. On the other hand, neural decoders learn this mapping from simulated shots and show comparable performance to algorithmic decoders on both surface and qLDPC codes, either end-to-end, mapping the syndrome straight to the logical outcome [45, 2, 27, 3], or by learning the message passing itself and reading faults off its marginals [31, 20, 32]. Their output, however, is taken on trust: a logical label cannot be checked against the syndrome at all, and a standalone learned message-passing decoder commits its fault pattern whether or not it reproduces the syndrome, so a wrong prediction passes without a second line of defence. And because one network must cover the hard tail of the fault distribution as well as the easy majority, this accuracy costs large recurrent transformers trained on 10^8–10^9 shots [2, 3] or many learned rounds over the entire decoding graph [32], paid on every shot, while the classical decoder is discarded altogether.

The two families are complementary. Algorithmic decoders provide the search that resolves the hard shots but carry nothing from one shot to the next; neural decoders learn from data but cannot be verified, and they must grow large enough to cover the hard tail. This suggests keeping the classical decoder and placing a learned model in front of it, provided the model is (i) **cheap**: its per-shot cost scales with the fired detectors rather than with the size of the decoding graph; (ii) **checkable**: a shot it settles on its own passes an explicit syndrome test, and the logical risk of such shots can be bounded; and (iii) **guiding**: the shots it cannot settle still receive what it has learned, in a form the backend can use. Requirements (ii) and (iii) rule out a logical label as output. The model must instead estimate mechanism-level fault posteriors on the detector error model (DEM) from which every backend is built, so that a thresholded candidate can be checked against the observed syndrome and the soft estimates can be handed to any DEM-based decoder.

Accordingly, we propose **CoD (Commit-or-Defer)**, a checkable neural pre-decoding framework that couples a compact fault-posterior network with a zero-residual commit rule and a posterior handoff, which is unified across codes and backends. To keep it cheap, the network runs bidirectional GRU message passing only on a K-hop active subgraph of the DEM bipartite graph around the fired detectors, reading out each fault probability as a correction to its prior log-odds; its cost thus grows with the fired detectors rather than with the decoding graph, and one model per instance, trained on about 10^7 shots, covers all physical error rates.. To make commitment checkable, CoD thresholds the posteriors into a candidate and commits only when the syndrome residual on the full DEM vanishes. Syndrome consistency forces any committed logical error to be an undetectable discrepancy of weight at least the DEM logical distance, which bounds the logical risk of committed shots by posterior uncertainty and calibration error relative to that distance and yields a sufficient condition for lower overall risk than the unassisted backend. To guide deferred shots, the neural posteriors are blended into the backend's priors or condition its search on the most contested mechanism, so patterns learned from data enter the classical inference instead of being re-solved from scratch. Consequently, relative to algorithmic decoding, CoD bypasses the backend on most shots and informs it on the rest; relative to standalone neural decoding, every committed candidate has been checked, and the network stays small because it need not cover the tail. In summary, our contributions are as follows.

1) A checkable neural pre-decoding framework. We propose CoD, in which a compact fault-posterior network, a zero-residual commit rule and a posterior handoff let a learned model precede a classical decoder without being taken on trust: committed shots bypass the backend after a syndrome check, and deferred shots carry the same posteriors into it.

2) A prior-anchored posterior network on active DEM subgraphs. We design the network around K-hop active subgraphs of the DEM bipartite graph, bidirectional GRU message passing with sum–max aggregation shared across rounds, a readout that corrects the physical prior log-odds, and a reweighted objective that preserves posterior calibration, so that its cost grows with the fired detectors rather than with the code, its estimates remain on the scale of the priors for blending, and one model per instance covers the full error-rate range.

3) Logical-risk analysis of commit-or-defer. We show that a committed logical error must be an undetectable discrepancy of at least $d_{\log}$ mechanisms, which bounds the logical risk of committed shots and yields a sufficient condition under which CoD lowers overall logical risk below the unassisted backend.

4) Data-efficient decoding at low neural cost. With approximately 10^7 training shots per circuit-level model, CoD lowers mean LER by up to 42.8% and backend calls by 55–93% across ten instances spanning four code families and three backends. Under code-capacity noise on BB72 and BB144,

both assisted decoders achieve lower mean LER than a standard graph neural decoder using roughly three million training shots and 42–56× fewer neural inference FLOPs.

2 PRELIMINARIES

2.1 QUANTUM ERROR CORRECTION AND DETECTOR ERROR MODELS

Stabilizer codes. Quantum error correction protects k logical qubits by encoding them into n physical qubits. A stabilizer code specifies the valid encoded states through commuting Pauli operators, called stabilizers [23, 10]. Formally, let $\mathcal{S}$ be an Abelian subgroup of the n-qubit Pauli group, with $-I \notin \mathcal{S}$ and $n-k$ independent generators, where I is the identity operator. An encoded state $|\psi\rangle$ is fixed by every stabilizer $S \in \mathcal{S}$: $S|\psi\rangle = |\psi\rangle, \forall S \in \mathcal{S}$. These states form a 2^k-dimensional code space. If a Pauli error E anticommutes with S, then $S(E|\psi\rangle) = -E|\psi\rangle$. Measuring S therefore changes its outcome from $+1$ to -1, revealing evidence of the error. The code is denoted $[[n, k, d]]$, where d is the minimum weight of a Pauli operator that commutes with every stabilizer but is not a stabilizer up to global phase. The codes considered here are Calderbank–Shor–Steane (CSS) codes, whose generators separate into X-type and Z-type checks [9, 42].

Memory experiments. A memory experiment tests recovery of encoded logical information after repeated noisy syndrome extraction [21]. A Z-memory experiment prepares k logical qubits in $|0\rangle_L^{\otimes k}$, performs T stabilizer-measurement rounds, then measures the data qubits in the Z basis; their parities give the k logical Z outcomes. An X-memory experiment instead prepares $|+\rangle_L^{\otimes k}$ and uses X-basis readout. Preparation, gate and measurement errors can alter the record. A *detector* checks a recorded parity that is deterministic without noise and fires when it differs from its reference [17]; the parity of consecutive measurements of the same stabilizer, for example, detects changes between rounds. The D outcomes form the *detector syndrome* $\boldsymbol{s} \in \mathbb{F}_2^D$, hereafter the syndrome, over the binary field $\mathbb{F}_2$, with $s_i = 1$ when detector $i \in \{1, \dots, D\}$ fires. Let $\boldsymbol{y}^{\text{logical}} \in \mathbb{F}_2^k$ record logical readout flips relative to the prepared values. From $\boldsymbol{s}$, the decoder predicts $\hat{\boldsymbol{y}}^{\text{logical}} \in \mathbb{F}_2^k$; correcting the readout recovers the stored logical values exactly when $\hat{\boldsymbol{y}}^{\text{logical}} = \boldsymbol{y}^{\text{logical}}$.

Detector error models. A detector error model (DEM) describes how circuit errors produce syndrome bits and logical readout flips by assigning each mechanism an occurrence probability and the detectors and logical observables it flips. These effects follow from propagating errors through the measurement circuit [17].

For the independent-mechanism DEM used here, let $\boldsymbol{y}^{\text{fault}} \in \mathbb{F}_2^m$ indicate which of the m mechanisms occur, indexed by $j = 1, \dots, m$. The vector $\boldsymbol{\pi} = (\pi_j)_{j=1}^m \in (0,1)^m$ contains their prior occurrence probabilities before the syndrome is observed, as specified by the circuit noise model and its physical error-rate parameter p. The fault indicators are sampled independently: $y_j^{\text{fault}} \overset{\text{ind}}{\sim} \text{Bernoulli}(\pi_j)$. Define $\boldsymbol{H} \in \mathbb{F}_2^{D \times m}$ and $\boldsymbol{L} \in \mathbb{F}_2^{k \times m}$, where $H_{ij} = 1$ if mechanism j flips detector i, and $L_{aj} = 1$ if it flips logical observable $a \in \{1, \dots, k\}$. The resulting syndrome and logical labels are

$$\boldsymbol{s} = \boldsymbol{H}\boldsymbol{y}^{\text{fault}}, \qquad \boldsymbol{y}^{\text{logical}} = \boldsymbol{L}\boldsymbol{y}^{\text{fault}}. \tag{1}$$

The products are over $\mathbb{F}_2$, so repeated flips cancel in pairs. Together, $\boldsymbol{H}, \boldsymbol{L}, \boldsymbol{\pi}$ specify the model.

2.2 DECODING

With $\boldsymbol{H}, \boldsymbol{L}$ fixed, a decoder $f : \mathbb{F}_2^D \times (0,1)^m \to \mathbb{F}_2^k$ predicts logical flips from the syndrome and priors. It can predict them directly or first estimate a fault candidate $\hat{\boldsymbol{y}}^{\text{fault}} \in \mathbb{F}_2^m$ and return

$$f(\boldsymbol{s}, \boldsymbol{\pi}) = \boldsymbol{L}\hat{\boldsymbol{y}}^{\text{fault}}. \tag{2}$$

Probabilistic fault prediction estimates posterior marginals $q_j \approx p_j(\boldsymbol{s}) = \Pr[y_j^{\text{fault}} = 1 \mid \boldsymbol{s}]$, collected in $\boldsymbol{q} \in [0,1]^m$, and thresholds them as $\hat{y}_j^{\text{fault}} = \mathbf{1}[q_j \geq \frac{1}{2}]$, where $\mathbf{1}[\cdot]$ is the indicator. The candidate is syndrome-consistent when $\boldsymbol{H}\hat{\boldsymbol{y}}^{\text{fault}} = \boldsymbol{s}$; the soft probabilities can also guide an algorithmic backend. Both strategies admit algorithmic or learned decoders (Appendix A); $\boldsymbol{\pi}$ is fixed across shots under the same noise model.

3 COMMIT-OR-DEFER PRE-DECODING (COD)

CoD (Commit-or-Defer) is a data-efficient neural pre-decoder that estimates fault probabilities $q_j \approx p_j(\boldsymbol{s})$ with a lightweight network on a compact active subgraph (Section 3.1). These estimates

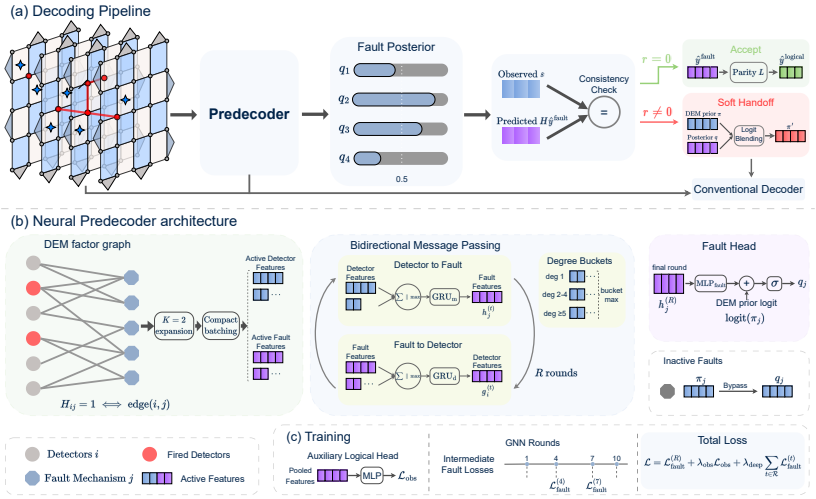


Figure 1: Overview of CoD. A network estimates fault posteriors on the active DEM subgraph. Syndrome-consistent candidates are committed; other shots are passed to a classical decoding procedure guided by the posterior.

support two decisions, shown in Fig. 1: *commit* a syndrome-consistent candidate without invoking the backend (Section 3.2), or *defer* and use the soft probabilities to guide the backend (Section 3.3). Even when the hard candidate fails the syndrome check, its soft estimates can distinguish competing fault configurations. Section 3.4 describes how the model is trained for this strategy.

3.1 Pre-Decoder Architecture

Given syndrome $\boldsymbol{s}$ and DEM $(\boldsymbol{H}, \boldsymbol{L}, \boldsymbol{\pi})$, the neural pre-decoder constructs an active bipartite graph, encodes detector and mechanism features, and passes recurrent messages between the two node types. A shared head maps final mechanism states to fault probabilities.

Input Graph. The DEM defines a bipartite graph with one node per detector and one node per fault mechanism. An edge connects detector node i and mechanism node j whenever $H_{ij} = 1$, giving the neighbourhoods $N_{\mathrm{D}}(j) = \{i : H_{ij} = 1\}, N_{\mathrm{M}}(i) = \{j : H_{ij} = 1\}$. Each mechanism node connects to every detector affected by the corresponding fault.

Detector node features. For each detector node i, let τ_i be its recorded time layer divided by T, and let c_i be the supplied check-type indicator: 1 for a recorded X-check and 0 for a recorded Z-check or an unannotated detector. Its four-dimensional input vector is

$$\boldsymbol{x}_i^{\mathrm{d}} = \left[s_i,\ \tau_i,\ c_i,\ \log(1 + |N_{\mathrm{M}}(i)|)\right]^\top \in \mathbb{R}^4. \tag{3}$$

These features combine the observed syndrome bit with the detector's time layer, check-type metadata, and connectivity in the full DEM graph.

Mechanism node features. Let $\alpha_\pi > 0$ be a fixed prior-feature scaling coefficient. The logit transform gives the prior log-odds $\operatorname{logit} \pi_j = \log[\pi_j/(1 - \pi_j)]$. Using the k entries of column j of $\boldsymbol{L}$, each mechanism node j carries

$$\boldsymbol{x}_j^{\mathrm{m}} = \left[\alpha_\pi \operatorname{logit} \pi_j,\ \log(1 + |N_{\mathrm{D}}(j)|),\ L_{1j}, \ldots, L_{kj}\right]^\top \in \mathbb{R}^{k+2}. \tag{4}$$

The scaled log-odds encode prior occurrence probability, the degree counts affected detectors, and the final k bits identify flipped logical observables. Both feature vectors represent binary entries as real-valued zeros and ones. Detector features thus encode observed evidence, and mechanism features describe possible fault explanations.

Encoders. Let d_{h} be the common hidden width and R the number of message-passing rounds. We write $\boldsymbol{h}_j^{(t)}, \boldsymbol{g}_i^{(t)} \in \mathbb{R}^{d_{\mathrm{h}}}$ for mechanism and detector states after round $t = 0, \ldots, R$, with $t = 0$ denoting the encoder outputs. The affine encoders have learned weights $\boldsymbol{E}_{\mathrm{m}} \in \mathbb{R}^{d_{\mathrm{h}} \times (k+2)}$, $\boldsymbol{E}_{\mathrm{d}} \in \mathbb{R}^{d_{\mathrm{h}} \times 4}$ and biases $\boldsymbol{e}_{\mathrm{m}}, \boldsymbol{e}_{\mathrm{d}} \in \mathbb{R}^{d_{\mathrm{h}}}$, and initialize the states as

$$\boldsymbol{h}_j^{(0)} = \boldsymbol{E}_{\mathrm{m}} \boldsymbol{x}_j^{\mathrm{m}} + \boldsymbol{e}_{\mathrm{m}}, \qquad \boldsymbol{g}_i^{(0)} = \boldsymbol{E}_{\mathrm{d}} \boldsymbol{x}_i^{\mathrm{d}} + \boldsymbol{e}_{\mathrm{d}}. \tag{5}$$

Message passing on the active subgraph. Full-graph neural decoding propagates syndrome information through repeated message passing [32], with FLOP cost growing with graph size and the number of rounds. At low physical error rates, faults typically form small, separated clusters [26], motivating computation near fired detectors.

Active subgraph. Starting from the fired-detector set $\mathcal{D}_0(\boldsymbol{s}) = \{i : s_i = 1\}$, we expand the subgraph for K steps. Each detector–mechanism–detector expansion counts as one hop (two bipartite edges), giving

$$\begin{aligned} \mathcal{M}_h(\boldsymbol{s}) &= N_{\mathrm{M}}(\mathcal{D}_{h-1}(\boldsymbol{s})), \\ \mathcal{D}_h(\boldsymbol{s}) &= \mathcal{D}_{h-1}(\boldsymbol{s}) \cup N_{\mathrm{D}}(\mathcal{M}_h(\boldsymbol{s})), \qquad h = 1, \dots, K, \end{aligned} \tag{6}$$

where neighbourhoods of sets are unions of node neighbourhoods. The active subgraph contains $\mathcal{M}_K(\boldsymbol{s})$, all adjacent detectors, and the DEM edges between them. Their observed syndrome bits retain odd-parity evidence from fired detectors and even-parity evidence from non-fired detectors. We encode the nodes and perform R message-passing rounds on this fixed subgraph.

Sum–max message aggregation. Detector syndromes constrain the joint parity of neighbouring faults. To capture both cumulative evidence and the strongest response in each feature, agg concatenates the sum and coordinatewise maximum of neighbouring states before each node update. For a mechanism node at round $t \geq 1$,

$$\mathrm{agg}_{i \in N_{\mathrm{D}}(j)} \boldsymbol{g}_i^{(t-1)} = \left[\sum_{i \in N_{\mathrm{D}}(j)} \boldsymbol{g}_i^{(t-1)} \,\middle\|\, \max_{i \in N_{\mathrm{D}}(j)} \boldsymbol{g}_i^{(t-1)} \right] \in \mathbb{R}^{2d_{\mathrm{h}}}. \tag{7}$$

The same operator aggregates mechanism states at detectors. Learned projections combine the summaries before each update. Maxima are computed in degree buckets by batched reduction over padded neighbour states, with padding masked out.

GRU state updates. Gated recurrent units let each node retain previous information or revise it using new messages [11]. At each round $t = 1, \dots, R$, $\mathrm{GRU}_{\mathrm{m}}$ first updates mechanism states from detector states, then $\mathrm{GRU}_{\mathrm{d}}$ updates detector states from the new mechanism states. Each update combines projected messages, original features and a bias with the previous state:

$$\begin{aligned} \boldsymbol{h}_j^{(t)} &= \mathrm{GRU}_{\mathrm{m}}\Big(\boldsymbol{W}_{\mathrm{m}} \operatorname*{agg}_{i \in N_{\mathrm{D}}(j)} \boldsymbol{g}_i^{(t-1)} + \boldsymbol{U}_{\mathrm{m}} \boldsymbol{x}_j^{\mathrm{m}} + \boldsymbol{b}_{\mathrm{m}},\ \boldsymbol{h}_j^{(t-1)}\Big), \\ \boldsymbol{g}_i^{(t)} &= \mathrm{GRU}_{\mathrm{d}}\Big(\boldsymbol{W}_{\mathrm{d}} \operatorname*{agg}_{j \in N_{\mathrm{M}}(i) \cap \mathcal{M}_K(\boldsymbol{s})} \boldsymbol{h}_j^{(t)} + \boldsymbol{U}_{\mathrm{d}} \boldsymbol{x}_i^{\mathrm{d}} + \boldsymbol{b}_{\mathrm{d}},\ \boldsymbol{g}_i^{(t-1)}\Big). \end{aligned} \tag{8}$$

The learned message projections are $\boldsymbol{W}_{\mathrm{m}}, \boldsymbol{W}_{\mathrm{d}} \in \mathbb{R}^{d_{\mathrm{h}} \times 2d_{\mathrm{h}}}$, feature projections are $\boldsymbol{U}_{\mathrm{m}} \in \mathbb{R}^{d_{\mathrm{h}} \times (k+2)}$ and $\boldsymbol{U}_{\mathrm{d}} \in \mathbb{R}^{d_{\mathrm{h}} \times 4}$, and biases are $\boldsymbol{b}_{\mathrm{m}}, \boldsymbol{b}_{\mathrm{d}} \in \mathbb{R}^{d_{\mathrm{h}}}$. To keep the model compact, all update parameters are shared across rounds.

Fault-probability readout. Let $\mathrm{MLP}_{\mathrm{fault}}$ be a shared two-layer MLP with a scalar output and a SiLU activation between its layers, and let σ denote the sigmoid. We compute posterior probability estimates from the final states:

$$q_j = \begin{cases} \sigma\Big(\operatorname{logit} \pi_j + \mathrm{MLP}_{\mathrm{fault}}(\boldsymbol{h}_j^{(R)})\Big), & j \in \mathcal{M}_K(\boldsymbol{s}), \\ \pi_j, & j \notin \mathcal{M}_K(\boldsymbol{s}). \end{cases} \tag{9}$$

The head learns a syndrome-dependent correction to the prior log-odds; the sigmoid maps their sum to a posterior probability estimate. Inactive mechanisms retain their physical priors. The complete vector $\boldsymbol{q}$ supplies the commitment and handoff rules below.

3.2 Zero-Residual Commit

Commit Policy. We obtain the fault candidate $\hat{\boldsymbol{y}}^{\mathrm{fault}}$ using the threshold rule in Section 2.2. Since thresholding need not satisfy the detector parity constraints, we compute the *syndrome residual*,

$$\boldsymbol{r} = \boldsymbol{s} \oplus \boldsymbol{H}\hat{\boldsymbol{y}}^{\mathrm{fault}} \in \mathbb{F}_2^D, \tag{10}$$

where $\oplus$ denotes componentwise addition modulo two. The check uses the full DEM. If $\boldsymbol{r} = \boldsymbol{0}$, we *commit* and return $\hat{\boldsymbol{y}}^{\mathrm{logical}} = \boldsymbol{L}\hat{\boldsymbol{y}}^{\mathrm{fault}}$ without invoking the backend. Otherwise, we *defer* the shot

to a classical backend, using the neural probability estimates $\boldsymbol{q}$ to guide its decoding of the original syndrome.

Logical risk after commit. For a committed prediction in $\mathcal{E}_{\boldsymbol{s}} = \{\boldsymbol{u} \in \mathbb{F}_2^m : \boldsymbol{H}\boldsymbol{u} = \boldsymbol{s}\}$, the discrepancy $\boldsymbol{z} = \boldsymbol{y}^{\text{fault}} \oplus \hat{\boldsymbol{y}}^{\text{fault}}$ satisfies $\boldsymbol{H}\boldsymbol{z} = \boldsymbol{0}$, and a logical failure can still occur when $\boldsymbol{L}\boldsymbol{z} \neq \boldsymbol{0}$. We define the DEM logical distance as the minimum number of mechanisms in such an undetectable logical discrepancy:

$$d_{\log} = \min\{|\boldsymbol{z}| : \boldsymbol{z} \in \mathbb{F}_2^m,\ \boldsymbol{H}\boldsymbol{z} = \boldsymbol{0},\ \boldsymbol{L}\boldsymbol{z} \neq \boldsymbol{0}\}, \tag{11}$$

where $|\boldsymbol{z}|$ denotes Hamming weight [33]. The distance $d_{\log}$ counts mechanisms, whereas the code distance d counts physical-qubit support. We set $d_{\log} = +\infty$ if the set is empty. Every logical failure after commitment therefore requires at least $d_{\log}$ incorrectly predicted mechanisms.

The expected number of incorrectly predicted mechanisms satisfies $\mathbb{E}[|\boldsymbol{z}| \mid \boldsymbol{s}] \leq S_{\boldsymbol{q}}(\boldsymbol{s}) + \varepsilon(\boldsymbol{s})$, where

$$S_{\boldsymbol{q}}(\boldsymbol{s}) = \sum_{j=1}^{m} \min\{q_j, 1 - q_j\}, \qquad \varepsilon(\boldsymbol{s}) = \sum_{j=1}^{m} |q_j - p_j(\boldsymbol{s})|. \tag{12}$$

Here $S_{\boldsymbol{q}}(\boldsymbol{s})$ measures uncertainty in the predicted fault probabilities, while $\varepsilon(\boldsymbol{s})$ measures their total deviation from the true posterior marginals. Appendix B.2 provides the full derivation.

Improvement over the unassisted backend. Let $\hat{\boldsymbol{y}}_{\mathrm{B}}^{\text{logical}}(\boldsymbol{s})$ and $\hat{\boldsymbol{y}}_{\mathrm{G}}^{\text{logical}}(\boldsymbol{s})$ denote the logical predictions of the unassisted backend and the guided backend of Section 3.3, respectively. Define the commit event and its subset of logical disagreements with the unassisted backend by

$$\mathcal{C} = \{\boldsymbol{r} = \boldsymbol{0}\}, \qquad \mathcal{A} = \mathcal{C} \cap \{\boldsymbol{L}\hat{\boldsymbol{y}}^{\text{fault}}(\boldsymbol{s}) \neq \hat{\boldsymbol{y}}_{\mathrm{B}}^{\text{logical}}(\boldsymbol{s})\}. \tag{13}$$

Only disagreements can change logical accuracy. Their probabilities over all shots are

$$\alpha = \Pr(\mathcal{A}), \qquad \zeta = \Pr\Big(\mathcal{C}^c \cap \{\hat{\boldsymbol{y}}_{\mathrm{G}}^{\text{logical}}(\boldsymbol{s}) \neq \hat{\boldsymbol{y}}_{\mathrm{B}}^{\text{logical}}(\boldsymbol{s})\}\Big). \tag{14}$$

Here α and ζ are the probabilities of committed and deferred disagreements, respectively.

For $\alpha > 0$, let $\varepsilon_A \geq 0$ be a prescribed finite budget for the mean posterior-estimation error on $\mathcal{A}$, and define

$$u_A = \min\left\{1, \frac{\mathbb{E}[S_{\boldsymbol{q}}(\boldsymbol{s}) \mid \mathcal{A}] + \varepsilon_A}{d_{\log}}\right\}, \tag{15}$$

with $u_A = 0$ when $d_{\log} = +\infty$. Under the posterior-accuracy condition in Appendix B.1, u_A bounds the neural logical risk on $\mathcal{A}$. Let R_{CoD} and R_{B} denote the overall logical failure probabilities of CoD and the unassisted backend under the same DEM, respectively.

Proposition 1 (Logical-risk improvement)**.** *Under Assumption 1 in Appendix B.1, CoD has strictly lower overall logical risk than the unassisted backend:*

$$R_{\mathrm{CoD}} - R_{\mathrm{B}} \leq -\alpha(1 - 2u_A) + \zeta < 0. \tag{16}$$

Appendix B.2 gives the proof.

3.3 Backend Handoff

On deferred shots $\mathcal{C}^c$, CoD invokes a classical backend on the original syndrome $\boldsymbol{s}$ and returns $\hat{\boldsymbol{y}}^{\text{logical}} = \hat{\boldsymbol{y}}_{\mathrm{G}}^{\text{logical}}(\boldsymbol{s})$. Although the neural hard prediction fails the consistency check, its individual probability estimates $\boldsymbol{q}$ can still provide useful guidance. We use these estimates through prior blending or conditioned search, depending on the backend.

Prior blending. For BP-OSD and MWPM, we form modified backend-input probabilities $\boldsymbol{\pi}'$ with a guidance coefficient $\beta \in [0, 1]$:

$$\operatorname{logit} \pi'_j = (1 - \beta) \operatorname{logit} \pi_j + \beta \operatorname{logit} q_j. \tag{17}$$

Setting $\beta = 0$ retains the physical priors, $\beta = 1$ uses the neural estimates, and intermediate values apply a partial correction. The backend decodes the original syndrome using these modified probabilities.

Conditioned search. Alternatively, the guided backend first obtains an initial fault candidate $\hat{\boldsymbol{y}}^{\text{fault,base}}$ using $\boldsymbol{\pi}$. Among mechanisms for which $\mathbf{1}[q_j > 1/2] \neq \hat{y}_j^{\text{fault,base}}$, we select $j_\star$ with the

largest absolute change in log-odds from the physical prior, provided that this change exceeds a numerical tolerance. The strict inequality resolves ties to zero within this search. We write $j_\star = \varnothing$ when none is eligible. Otherwise, fixing the selected mechanism to $b = \mathbf{1}[q_{j_\star} > 1/2]$ restricts the search to $\{\boldsymbol{u} \in \mathcal{E}_{\boldsymbol{s}} : u_{j_\star} = b\}$. The additional backend run targets the equivalent reduced constraint

$$\boldsymbol{H}_{-j_\star}\boldsymbol{u}_{-j_\star} = \boldsymbol{s} \oplus \boldsymbol{H}_{:j_\star} b \tag{18}$$

using the original priors $\boldsymbol{\pi}_{-j_\star}$. Here $\boldsymbol{H}_{:j_\star}$ is column $j_\star$, and the subscript $-j_\star$ removes the selected column or component. Restoring the fixed bit gives a candidate $\hat{\boldsymbol{y}}^{\text{fault,cond}}$.

To compare the candidates, let $P_{\boldsymbol{\pi}}(\boldsymbol{u}) = \prod_{j=1}^{m} \pi_j^{u_j}(1-\pi_j)^{1-u_j}$ be the probability of $\boldsymbol{u}$ under the independent-mechanism DEM. The guided logical prediction is

$$\hat{\boldsymbol{y}}_{\mathrm{G}}^{\text{logical}}(\boldsymbol{s}) = \begin{cases} \boldsymbol{L}\hat{\boldsymbol{y}}^{\text{fault,cond}}, & \text{if } j_\star \neq \varnothing, \quad \hat{\boldsymbol{y}}^{\text{fault,cond}} \in \mathcal{E}_{\boldsymbol{s}}, \\ & \text{and } \left[\hat{\boldsymbol{y}}^{\text{fault,base}} \notin \mathcal{E}_{\boldsymbol{s}} \vee P_{\boldsymbol{\pi}}(\hat{\boldsymbol{y}}^{\text{fault,cond}}) > P_{\boldsymbol{\pi}}(\hat{\boldsymbol{y}}^{\text{fault,base}})\right], \\ \boldsymbol{L}\hat{\boldsymbol{y}}^{\text{fault,base}}, & \text{otherwise.} \end{cases} \tag{19}$$

3.4 Training Objective

Overall, with coefficients λ_{obs} for the auxiliary logical loss and λ_{deep} for the intermediate fault losses, the total objective $\mathcal{L}$ is

$$\mathcal{L} = \mathcal{L}_{\text{fault}}^{(R)} + \lambda_{\text{obs}}\,\mathcal{L}_{\text{obs}} + \lambda_{\text{deep}} \sum_{t \in \mathcal{R}} \mathcal{L}_{\text{fault}}^{(t)}, \tag{20}$$

where $\mathcal{L}_{\text{fault}}^{(t)}$ is the corrected reweighted binary cross-entropy for fault labels at round t, $\mathcal{L}_{\text{obs}}$ is the auxiliary logical-label cross-entropy, and $\mathcal{R} \subseteq \{1, \ldots, R-1\}$ contains the rounds used for deep supervision. The objective combines direct logical supervision with learning fault posteriors for commitment and backend guidance. Appendix B.3 explains the role and construction of each loss term.

4 Experiments

We evaluate CoD across code families and classical backends under circuit-level noise, compare its accuracy, data efficiency and neural inference cost with standalone neural decoding under code-capacity noise, and examine how active neighbourhoods and aggregation affect decoding performance.

4.1 Setup

Codes, memory experiments and noise. We evaluate ten instances from four families: bivariate bicycle codes BB72, BB108 and BB144 [7]; random quantum Tanner codes RQT60 and RQT108, using the benchmark instances released with PropHunt [46]; rotated surface codes at $d = 5, 7, 9$ [16]; and triangular 6.6.6 colour codes at $d = 5, 11$ [5]. Each runs $T = d$ syndrome-extraction rounds with noisy final readout under SI1000 noise [19]. Circuit constructions and memory bases are in Appendix C.1 (Tab. 3). The separate code-capacity comparison uses BB72 and BB144 with perfect syndrome measurements and independent categorical Pauli errors; noise probabilities and the test grid are in Appendix C.2.

Backends and baselines. We use *BP-OSD* [39] on bivariate bicycle, Tanner and colour codes, *Relay-BP* [34] on all four families, and *MWPM* [25] on surface codes; standalone *Astra* [32] is the code-capacity baseline on BB72 and BB144. Decoder and handoff parameters are in Appendix C.4.

Training and evaluation. Each circuit-level model uses approximately 10^7 online training shots sampled at a few error rates and is fixed across its full test grid, including unseen rates, with the corresponding DEM prior in Eq. (9). Code-capacity models are trained separately with roughly three million samples per learned model for each code (Appendix C.3). Methods within each comparison decode identical test shots, with frozen Pauli-error samples for code capacity. Shot counts, ablation test populations and uncertainty reporting are specified in Appendix C.6.

Metrics. We evaluate logical accuracy using LER (↓), backend use using Backend call rate (↓), and the additional BP-OSD benefit using OSD call reduction (↑) relative to pure BP-OSD; definitions and statistical conventions are in Appendix C.5 and Appendix C.6.

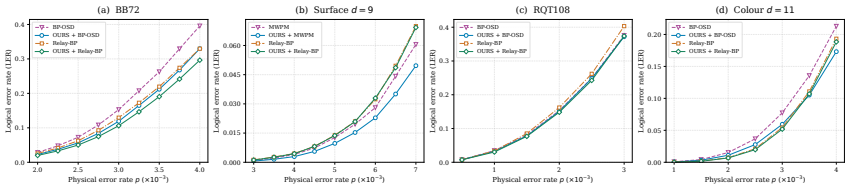


Figure 2: Logical error rate (↓) versus physical error rate on (a) BB72, (b) the surface code at $d = 9$, (c) RQT108 and (d) the colour code at $d = 11$ under SI1000 circuit-level noise. The remaining six instances are shown in Fig. 5.

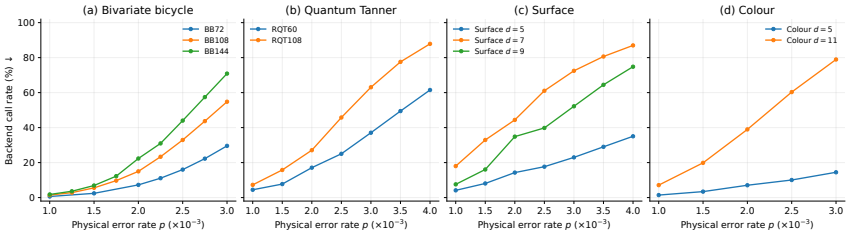


Figure 3: Backend call rate (↓) versus physical error rate under SI1000 circuit-level noise.

Table 1: Results at $p = 0.002$ under SI1000 noise. B is MWPM for surface codes and BP-OSD otherwise. Assisted results are mean ± sample SD on identical shots. ΔLER is relative to the corresponding pure backend. Metric definitions are in Appendix C.5.

	B	CoD+B				Relay-BP	CoD+Relay-BP		
Code	LER ($\times 10^{-2}$)	LER ($\times 10^{-2}$)	ΔLER (%)	Backend call rate (%) ↓	OSD call reduction (%) ↑	LER ($\times 10^{-2}$)	LER ($\times 10^{-2}$)	ΔLER (%)	Backend call rate (%) ↓
BB72	2.840	2.259±0.011	↓ 20.47	7.26±0.19	68.93±2.02	2.560	1.983±0.009	↓ 22.53	7.26±0.19
BB108	0.960	0.789±0.034	↓ 17.78	14.98±0.38	59.90±1.06	0.166	0.145±0.008	↓ 12.45	14.98±0.38
BB144	0.840	0.481±0.017	↓ 42.78	22.34±1.80	72.21±1.98	0.136	0.137±0.010	↑ 0.98	22.34±1.80
Surface $d = 5$	0.180	0.164±0.012	↓ 8.80	14.28±1.60	N/A	0.157	0.138±0.003	↓ 11.89	14.28±1.60
Surface $d = 7$	0.048	0.043±0.001	↓ 10.56	44.40±3.78	N/A	0.047	0.045±0.002	↓ 3.42	44.40±3.78
Surface $d = 9$	0.011	0.008±0.000	↓ 23.67	34.85±0.82	N/A	0.030	0.026±0.001	↓ 13.22	34.85±0.82
RQT60	1.648	1.676±0.020	↑ 1.70	17.09±2.25	70.07±1.98	1.710	1.691±0.016	↓ 1.13	17.09±2.25
RQT108	15.314	15.085±0.150	↓ 1.49	27.08±1.64	67.74±1.39	16.210	14.825±0.086	↓ 8.55	27.08±1.64
Colour $d = 5$	2.898	2.077±0.074	↓ 28.32	7.05±0.86	65.09±1.83	2.042	1.773±0.007	↓ 13.19	7.05±0.86
Colour $d = 11$	1.526	1.096±0.050	↓ 28.18	38.97±0.64	53.11±1.78	0.714	0.675±0.027	↓ 5.42	38.97±0.64

4.2 Circuit-Level Decoding Performance

Fig. 2 compares the LER curves of assisted and unassisted decoding across physical error rates for four representative code instances; Fig. 5 gives the remaining six. According to Tab. 1, CoD lowers the mean LER in **18 of the 20** code–backend configurations, including **nine of ten** instances for BP-OSD/MWPM and **nine of ten** for Relay-BP. Relative LER reductions reach 42.78% for BP-OSD/MWPM and 22.53% for Relay-BP. CoD achieves these gains while directly committing 55.60%–92.95% of shots. Fig. 3 shows that backend call rate increases with physical noise, with different trends across code instances. BP-OSD rows additionally report OSD call reduction.

4.3 Code-Capacity Accuracy and Neural Inference Cost

Tab. 2 reports the code-capacity LER results on BB72 and BB144; Fig. 4 relates these results to neural inference cost.

Accuracy gains from neural assistance. On BB144 ($d = 12$, $p = 0.08$), CoD lowers BP-OSD LER from 4.249% to 2.189% and Relay-BP LER from 1.291% to 1.032%, relative reductions of 48.5% and 20.1%, respectively (Fig. 4).

Neural inference cost. Both assisted decoders outperform Astra with 42–56× fewer neural inference FLOPs (Fig. 4). They share the same neural prediction and forward-pass cost; Appendix E.1 gives the analytical FLOP accounting.

Table 2: BB72 and BB144 code-capacity LER (%, ↓) at physical error rate p. Learned methods report the mean and sample standard deviation, with ± SD beneath each mean. Bold marks the lowest unrounded mean LER within each code and p, including ties. The bottom row gives the neural inference FLOP ratio F_{Astra}/F_{CoD} (↑), shared by both assisted backends.

Method	BB72									BB144								
p	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
BP-OSD	0.600	0.653	1.695	2.954	5.195	8.471	13.387	20.375	27.626	0.001	0.008	0.046	0.171	0.617	1.740	4.249	9.169	16.833
CoD+BP-OSD	0.072 ±0.004	0.331 ±0.004	0.821 ±0.007	1.916 ±0.025	3.720 ±0.005	6.686 ±0.022	10.786 ±0.065	16.640 ±0.024	23.381 ±0.017	0.001 ±0.001	0.001 ±0.000	0.012 ±0.002	0.056 ±0.003	0.262 ±0.003	0.801 ±0.016	2.189 ±0.044	5.084 ±0.093	10.042 ±0.105
Relay-BP	0.092	0.419	1.035	2.474	4.769	8.247	13.061	19.531	26.814	**0.000**†	**0.000**†	0.007	0.047	0.135	0.492	1.291	3.008	6.167
CoD+Relay-BP	**0.068 ±0.001**	**0.322 ±0.006**	**0.787 ±0.012**	**1.872 ±0.004**	**3.609 ±0.033**	**6.544 ±0.014**	**10.543 ±0.056**	**16.263 ±0.019**	**22.816 ±0.107**	**0.000**† **±0.000**	0.001 ±0.001	**0.004 ±0.001**	**0.026 ±0.002**	**0.100 ±0.007**	**0.343 ±0.007**	**1.032 ±0.019**	**2.590 ±0.020**	**5.480 ±0.042**
Astra	0.093 ±0.009	0.437 ±0.033	1.142 ±0.055	2.684 ±0.119	5.295 ±0.293	9.119 ±0.537	14.321 ±0.721	21.405 ±0.900	29.073 ±1.069	0.015 ±0.004	0.072 ±0.013	0.233 ±0.038	0.669 ±0.094	1.834 ±0.283	4.063 ±0.566	7.945 ±1.097	13.750 ±1.720	22.079 ±2.422
F_{Astra}/F_{CoD}	55.553×	47.670×	44.539×	43.177×	42.526×	42.243×	42.089×	42.037×	41.996×	47.738×	43.854×	42.634×	42.224×	42.073×	42.016×	41.997×	41.991×	41.988×

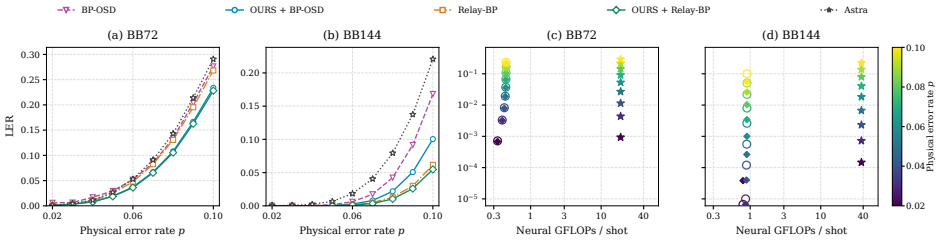


Figure 4: Code-capacity experiment. (a)–(b) System LER (↓) versus physical error rate on BB72 and BB144. (c)–(d) The same LERs versus neural GFLOPs per shot (↓); colour indicates p, circles denote CoD+BP-OSD, diamonds CoD+Relay-BP, and stars Astra. Accounting details appear in Appendix E.1; Tab. 2 reports the LER results.

Data efficiency. Roughly three million training shots per model suffice for both assisted decoders to achieve lower mean LERs than Astra at every tested p. On BB72 ($d = 6$, $p \in \{0.05, 0.08, 0.10\}$), extending Astra training to thirty million samples, about $10\times$ the data used by CoD, does not close the accuracy gap (Fig. 6).

4.4 Ablation Studies and Sensitivity Analysis

We analyse the effects of active-subgraph radius and message aggregation, the reliability of the commit rule, the benefit of backend guidance on deferred shots, and sensitivity to posterior-injection strength. These analyses examine the predictor, routing decision and posterior handoff in CoD.

The $K = 2$ active subgraph improves both accuracy and efficiency: on BB72 ($d = 6$), it lowers LER relative to the full graph at $p = 0.002$ and $p = 0.0025$, while accelerating training by $5.83\times$ and evaluation by 3.76–$5.14\times$ (Tab. 5). Sum–max aggregation gives the lowest measured LER and backend call rate and the highest OSD call reduction among the tested operators (Tab. 6). Appendix F.1 and Appendix F.2 give the architecture comparisons.

The commit analysis (Tab. 7) shows that the syndrome check concentrates most neural candidate logical errors in the deferred branch while bypassing the backend on most shots. On the same deferred subsets, guidance lowers mean LER for BP-OSD/MWPM on eight of ten instances and for Relay-BP on nine of ten (Tab. 8), showing that neural posteriors remain useful after the hard candidate fails the consistency check. On the colour code ($d = 11$, $p = 0.002$), the posterior-injection sweep isolates this guidance benefit by improving LER with routing held fixed (Fig. 7). Appendix F.3, Appendix F.4 and Appendix F.5 give the routing and guidance analyses.

5 Conclusion

In summary, we propose CoD, a cheap, checkable, and guiding neural predecoder for classical decoders. It uses local neural message passing, commits only after full-DEM syndrome verification, and guides deferred decoding with learned posteriors. With approximately 10^7 training shots per circuit-level model, it reduces mean LER by up to 42.8% and backend calls by 55–93% across ten instances, four code families and three backends. On BB72 and BB144 under code-capacity noise, it achieves lower mean LER than a standard graph neural decoder with roughly three million training shots and 42–$56\times$ fewer neural inference FLOPs. Future work will evaluate CoD on device-calibrated noise models that capture temporal drift and correlated errors.

AI USE STATEMENT

In this work, we used generative AI tools for drafting assistance and editing support during manuscript preparation. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts.

ETHICS STATEMENT

This work develops classical decoding algorithms for quantum error-correcting codes and evaluates them on simulated memory experiments. It involves no human subjects, personal data or hardware experiments, and we do not foresee direct ethical concerns; any societal impact is indirect, through progress towards fault-tolerant quantum computing.

REPRODUCIBILITY STATEMENT

The code instances, syndrome-extraction circuits and noise model are specified in Section 4.1 and Appendix C.1, the network in Section 3.1 and the training objective in Section 3.4. Training data and optimisation settings are listed in Appendix C.3, metric definitions in Appendix C.5, and test-shot counts, statistical conventions and provenance in Appendix C.6. Every comparison is evaluated on identical frozen test shots, and the ablation configurations are given in Appendix F.

REFERENCES

[1] Paul Baireuther, Thomas E. O'Brien, Brian Tarasinski, and Carlo W. J. Beenakker. Machine-learning-assisted correction of correlated qubit errors in a topological code. *Quantum*, 2:48, 2018.

[2] Johannes Bausch, Andrew W. Senior, Francisco J. H. Heras, Thomas Edlich, Alex Davies, Michael Newman, Cody Jones, Kevin Satzinger, Murphy Yuezhen Niu, Sam Blackwell, George Holland, Dvir Kafri, Juan Atalaya, Craig Gidney, Demis Hassabis, Sergio Boixo, Hartmut Neven, and Pushmeet Kohli. Learning high-accuracy error decoding for quantum processors. *Nature*, 635(8040):834–840, 2024.

[3] John Blue, Harshil Avlani, Zhiyang He, Liu Ziyin, and Isaac L. Chuang. Machine learning decoding of circuit-level noise for bivariate bicycle codes. *Quantum*, 10:2149, 2026.

[4] Dolev Bluvstein, Simon J. Evered, Alexandra A. Geim, Sophie H. Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, J. Pablo Bonilla Ataides, Nishad Maskara, Iris Cong, Xun Gao, Pedro Sales Rodriguez, Thomas Karolyshyn, Giulia Semeghini, Michael J. Gullans, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, 2024.

[5] Héctor Bombin and Miguel Angel Martin-Delgado. Topological quantum distillation. *Physical Review Letters*, 97(18):180501, 2006.

[6] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Physical Review A*, 90(3):032326, 2014.

[7] 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.

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

[9] A. Robert Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098–1105, 1996.

[10] A. Robert Calderbank, Eric M. Rains, Peter W. Shor, and N. J. A. Sloane. Quantum error correction via codes over GF(4). *IEEE Transactions on Information Theory*, 44(4):1369–1387, 1998.

[11] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

[12] Nicolas Delfosse. Decoding color codes by projection onto surface codes. *Physical Review A*, 89(1):012317, 2014.

[13] Nicolas Delfosse and Naomi H. Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, 2021.

[14] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[15] Guillaume Duclos-Cianci and David Poulin. Fast decoders for topological quantum codes. *Physical Review Letters*, 104(5):050504, 2010.

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

[17] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, 2021.

[18] Craig Gidney and Cody Jones. New circuits and an open source decoder for the color code. *arXiv preprint arXiv:2312.08813*, 2023.

[19] Craig Gidney, Michael Newman, Austin Fowler, and Michael Broughton. A fault-tolerant honeycomb memory. *Quantum*, 5:605, 2021.

[20] Anqi Gong, Sebastian Cammerer, and Joseph M. Renes. Graph neural networks for enhanced decoding of quantum LDPC codes. In *2024 IEEE International Symposium on Information Theory (ISIT)*, pp. 2700–2705, 2024.

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

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

[23] Daniel Gottesman. Class of quantum error-correcting codes saturating the quantum Hamming bound. *Physical Review A*, 54(3):1862–1868, 1996.

[24] N. Alan Heckert, James J. Filliben, C. M. Croarkin, B. Hembree, William F. Guthrie, P. Tobias, and J. Prinz. *NIST/SEMATECH e-Handbook of Statistical Methods*. National Institute of Standards and Technology, 2002. Section 1.3.5.6: Measures of Scale.

[25] Oscar Higgott and Craig Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. *Quantum*, 9:1600, 2025.

[26] Timo Hillmann, Lucas Berent, Armanda O. Quintavalle, Jens Eisert, Robert Wille, and Joschka Roffe. Localized statistics decoding for quantum low-density parity-check codes. *Nature Communications*, 16:8214, 2025.

[27] Gengyuan Hu, Wanli Ouyang, Chao-Yang Lu, Chen Lin, and Han-Sen Zhong. Efficient and universal neural-network decoder for stabilizer-based quantum error correction. *arXiv preprint arXiv:2502.19971*, 2025.

[28] Stefan Krastanov and Liang Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific Reports*, 7:11003, 2017.

[29] Kao-Yueh Kuo and Ching-Yi Lai. Exploiting degeneracy in belief propagation decoding of quantum codes. *npj Quantum Information*, 8:111, 2022.

[30] Moritz Lange, Pontus Havström, Basudha Srivastava, Isak Bengtsson, Valdemar Bergentall, Karl Hammar, Olivia Heuts, Evert van Nieuwenburg, and Mats Granath. Data-driven decoding of quantum error correcting codes using graph neural networks. *Physical Review Research*, 7 (2):023181, 2025.

[31] Ye-Hua Liu and David Poulin. Neural belief-propagation decoders for quantum error-correcting codes. *Physical Review Letters*, 122(20):200501, 2019.

[32] Arshpreet Singh Maan and Alexandru Paler. Machine learning message-passing for the scalable decoding of QLDPC codes. *npj Quantum Information*, 11:78, 2025.

[33] F. Jessie MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[34] Tristan Müller, Thomas Alexander, Michael E. Beverland, Markus Bühler, Blake R. Johnson, Thilo Maurer, and Drew Vandeth. Improved belief propagation is sufficient for real-time decoding of quantum memory. *arXiv preprint arXiv:2506.01779*, 2025.

[35] Vukan Ninkovic, Ognjen Kundacina, Dejan Vukobratovic, Christian Häger, and Alexandre Graell i Amat. Decoding quantum LDPC codes using graph neural networks. In *GLOBECOM 2024—2024 IEEE Global Communications Conference*, pp. 3479–3484, 2024.

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

[37] David Poulin and Yeojin Chung. On the iterative decoding of sparse quantum codes. *Quantum Information and Computation*, 8(10):987–1000, 2008.

[38] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018.

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

[40] Andrew W. Senior, Thomas Edlich, Francisco J. H. Heras, Lei M. Zhang, Oscar Higgott, James S. Spencer, Taylor Applebaum, Sam Blackwell, Justin Ledford, Akvilė Žemgulytė, Augustin Žídek, Noah Shutty, Andrew Cowie, Yin Li, George Holland, Peter Brooks, Charlie Beattie, Michael Newman, Alex Davies, Cody Jones, Sergio Boixo, Hartmut Neven, Pushmeet Kohli, and Johannes Bausch. A scalable and real-time neural decoder for topological quantum codes. *arXiv preprint arXiv:2512.07737*, 2025. Revised March 2026.

[41] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):R2493–R2496, 1995.

[42] Andrew M. 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, 1996.

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

[44] Giacomo Torlai and Roger G. Melko. Neural decoder for topological codes. *Physical Review Letters*, 119(3):030501, 2017.

[45] Savvas Varsamopoulos, Koen Bertels, and Carmen G. Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 69(2):300–311, 2020.

[46] 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.

[47] Stasiu Wolanski and Ben Barber. Ambiguity clustering: an accurate and efficient decoder for qLDPC codes. *arXiv preprint arXiv:2406.14527*, 2024.

A Related Work

A.1 Algorithmic Decoders

BP-OSD and minimum-weight perfect matching (MWPM) are representative baselines for qLDPC and surface-code decoding, respectively [36, 39, 14]. Their scope reflects the fault representation: BP-OSD operates on general parity constraints, whereas MWPM uses a graphlike model. For BB and other qLDPC codes, short cycles and degeneracy hinder BP convergence [37], leaving Gaussian elimination and candidate search to resolve difficult syndromes. LSD and ambiguity clustering reduce this cost by solving smaller reliability-guided clusters [26, 47]; Memory BP and Relay-BP instead modify message dynamics to escape trapping configurations [29, 34]. Their efficiency depends on cluster growth and iteration budgets, making difficult syndromes the main targets for reducing decoding cost.

Colour-code decoders exploit lattice structure to recover matching-based solutions: projection decoders map to three surface codes, and Chromobius uses Möbius matching [12, 18]. For surface codes, Sparse Blossom accelerates MWPM by searching the sparse detector graph directly [25], while union-find trades some matching accuracy for almost-linear cluster processing [13]. Renormalisation-group and tensor-network methods offer another trade-off, approximating logical inference through coarse graining or network contraction at adjustable computational cost [15, 6]. These methods benefit from code-specific geometry; their reductions do not directly provide a decoder for general BB fault models.

A.2 Neural Decoders

For qLDPC codes, Neural BP and Astra learn message updates on the Tanner graph, making parity-check connectivity part of the architecture [31, 32]. GNNs can also learn feedback between BP runs or replace the update rules with recurrent graph processing [20, 35]. This alignment supports reuse across code sizes, but inference still requires repeated neural computation. On BB codes, a code-aware recurrent transformer improves on BP-OSD for BB72 but loses that advantage on BB144 despite hundreds of millions of training examples [3]. GraphQEC combines graph-based and temporal inference across BB, colour and surface codes, while reporting rapidly growing data requirements with block length [27]. Thus encoding code structure in the network helps broaden applicability, but does not by itself resolve training and inference costs.

For surface codes, AlphaQubit is a representative approach based on recurrent transformers and experimental fine-tuning [2]. Like LSTM and detector-graph models [1, 30], it predicts logical outcomes from syndrome histories. This output formulation [45] avoids the repeated syndrome-compatible sampling of early probabilistic decoders [44, 28], but does not directly supply fault marginals for backend guidance. AlphaQubit 2 also reports real-time decoding for surface and colour codes, with a smaller model trading accuracy for speed [40]. Low latency is therefore achievable, but must be assessed alongside model capacity and training demand: AlphaQubit's reported pretraining uses hundreds of millions to billions of examples [2].

CoD uses a compact predictor on active DEM subgraphs to supply fault posteriors for both syndrome-checked commitment and backend guidance. Algorithmic search is then reserved for shots whose learned candidates fail the consistency check, making computation depend on whether the current syndrome has already been resolved.

B Theoretical Analysis of Commit-or-Defer

The risk guarantee in Section 3.2 follows from a bound on syndrome-consistent predictions and a decomposition of the commit and defer contributions.

B.1 Probabilistic Model and Assumptions

Fix the DEM, its priors, and the neural and backend parameters. We use $p_j(\boldsymbol{s})$ from Section 2.2 and $\mathcal{E}_{\boldsymbol{s}}$, $d_{\log}$, and the events $\mathcal{C}, \mathcal{A}$ from Section 3.2. For a syndrome $\boldsymbol{s}$ of positive probability, the neural estimate $\boldsymbol{q}(\boldsymbol{s})$ and the candidate $\hat{y}_j^{\mathrm{fault}}(\boldsymbol{s}) = \mathbf{1}[q_j(\boldsymbol{s}) \geq 1/2]$ are deterministic functions of $\boldsymbol{s}$. For

randomised backends, condition on algorithmic random choices independent of the true faults given $\boldsymbol{s}$; the same argument then averages over those choices, with $\mathcal{A}$ and the deferred-disagreement event including this randomness.

The DEM posterior.

$$\Pr[\boldsymbol{y}^{\text{fault}} = \boldsymbol{u} \mid \boldsymbol{s}] = \frac{\mathbf{1}[\boldsymbol{u} \in \mathcal{E}_{\boldsymbol{s}}] \prod_{j=1}^{m} \pi_j^{u_j} (1-\pi_j)^{1-u_j}}{\sum_{\boldsymbol{v} \in \mathcal{E}_{\boldsymbol{s}}} \prod_{j=1}^{m} \pi_j^{v_j} (1-\pi_j)^{1-v_j}}. \tag{21}$$

For a syndrome $\boldsymbol{s}$ of positive probability, this is the joint posterior under the independent-mechanism DEM. Selecting the most probable individual fault vector and selecting the logical class with the largest total posterior probability are different inference objectives. The fault marginals $p_j(\boldsymbol{s})$ describe individual mechanisms and generally do not determine the joint posterior.

Assumption 1 (Posterior accuracy and uncertainty margin)**.** Suppose $\alpha > 0$. The following conditions hold:

(i) **Posterior accuracy.** The mean estimation error on $\mathcal{A}$ satisfies

$$\mathbb{E}[\varepsilon(\boldsymbol{s}) \mid \mathcal{A}] \le \varepsilon_A. \tag{22}$$

(ii) **Low uncertainty with a margin for deferred changes.** The quantity in Eq. (15) satisfies

$$u_A < \frac{1}{2}\left(1 - \frac{\zeta}{\alpha}\right). \tag{23}$$

Condition (i) controls posterior estimation error on the committed shots whose logical predictions differ from the backend's. Condition (ii) requires their uncertainty and error budget to be small enough, relative to the DEM logical distance, to cover possible losses from changed deferred decisions.

B.2 Risk Improvement over the Unassisted Backend

Proof of Proposition 1. Write $e_{\mathrm{N}} = \mathbf{1}[\boldsymbol{L}\hat{\boldsymbol{y}}^{\text{fault}}(\boldsymbol{s}) \neq \boldsymbol{y}^{\text{logical}}]$, $e_{\mathrm{B}} = \mathbf{1}[\hat{\boldsymbol{y}}_{\mathrm{B}}^{\text{logical}}(\boldsymbol{s}) \neq \boldsymbol{y}^{\text{logical}}]$, and $e_{\mathrm{G}} = \mathbf{1}[\hat{\boldsymbol{y}}_{\mathrm{G}}^{\text{logical}}(\boldsymbol{s}) \neq \boldsymbol{y}^{\text{logical}}]$ for the three logical-error indicators.

Syndrome consistency bounds neural logical risk. First suppose $1 \le d_{\log} < \infty$. For a committed syndrome, both the true fault vector and the candidate satisfy $\boldsymbol{H}\boldsymbol{y}^{\text{fault}} = \boldsymbol{H}\hat{\boldsymbol{y}}^{\text{fault}} = \boldsymbol{s}$. Their discrepancy $\boldsymbol{z} = \boldsymbol{y}^{\text{fault}} \oplus \hat{\boldsymbol{y}}^{\text{fault}}$ therefore obeys $\boldsymbol{H}\boldsymbol{z} = \boldsymbol{0}$. If the logical decisions differ, then $\boldsymbol{L}\boldsymbol{z} \neq \boldsymbol{0}$, so the definition of $d_{\log}$ gives $|\boldsymbol{z}| \ge d_{\log}$. Consequently, pointwise on $\mathcal{E}_{\boldsymbol{s}}$,

$$e_{\mathrm{N}} \le \frac{|\boldsymbol{y}^{\text{fault}} \oplus \hat{\boldsymbol{y}}^{\text{fault}}|}{d_{\log}}. \tag{24}$$

Taking the conditional expectation gives

$$\mathbb{E}[e_{\mathrm{N}} \mid \boldsymbol{s}] \le \frac{\mu(\boldsymbol{s})}{d_{\log}}, \qquad \mu(\boldsymbol{s}) = \sum_{j=1}^{m} \big[(1-\hat{y}_j^{\text{fault}}) p_j(\boldsymbol{s}) + \hat{y}_j^{\text{fault}}(1-p_j(\boldsymbol{s}))\big]. \tag{25}$$

Here $\mu(\boldsymbol{s})$ is the conditional expected number of incorrectly predicted mechanisms. Linearity of expectation retains arbitrary dependencies between fault variables. Applying the same expression to $\boldsymbol{q}$ yields

$$\sum_j \big[(1-\hat{y}_j^{\text{fault}}) q_j + \hat{y}_j^{\text{fault}}(1-q_j)\big] = \sum_j \min\{q_j, 1-q_j\} = S_{\boldsymbol{q}}(\boldsymbol{s}). \tag{26}$$

Moreover,

$$\mu(\boldsymbol{s}) - S_{\boldsymbol{q}}(\boldsymbol{s}) = \sum_j (1 - 2\hat{y}_j^{\text{fault}})(p_j(\boldsymbol{s}) - q_j) \le \sum_j |p_j(\boldsymbol{s}) - q_j| = \varepsilon(\boldsymbol{s}). \tag{27}$$

Combining these expressions gives the pointwise bound

$$\Pr[\boldsymbol{L}\boldsymbol{y}^{\text{fault}} \neq \boldsymbol{L}\hat{\boldsymbol{y}}^{\text{fault}} \mid \boldsymbol{s}] \le \min\left\{1, \frac{S_{\boldsymbol{q}}(\boldsymbol{s}) + \varepsilon(\boldsymbol{s})}{d_{\log}}\right\}, \qquad \boldsymbol{s} \in \mathcal{C}. \tag{28}$$

Because $\mathcal{A} \subseteq \mathcal{C}$ is selected by the syndrome and decoder decisions, averaging and applying the posterior-error condition in Eq. (22) yield

$$\mathbb{E}[e_{\mathrm{N}} \mid \mathcal{A}] \leq \min\left\{1, \frac{\mathbb{E}[S_{\boldsymbol{q}}(\boldsymbol{s}) \mid \mathcal{A}] + \mathbb{E}[\varepsilon(\boldsymbol{s}) \mid \mathcal{A}]}{d_{\log}}\right\} \leq u_A. \tag{29}$$

If $d_{\log} = +\infty$, all feasible fault configurations for a given syndrome have the same logical label. A committed prediction is then logically correct, and Eq. (29) holds with $u_A = 0$.

The neural upper bound is below the backend lower bound. On $\mathcal{A}$, $\boldsymbol{L}\hat{\boldsymbol{y}}^{\mathrm{fault}}(\boldsymbol{s}) \neq \hat{\boldsymbol{y}}_{\mathrm{B}}^{\mathrm{logical}}(\boldsymbol{s})$, so these predictions cannot both equal $\boldsymbol{y}^{\mathrm{logical}}$. Therefore

$$e_{\mathrm{N}} + e_{\mathrm{B}} \geq 1, \qquad e_{\mathrm{N}} - e_{\mathrm{B}} \leq 2e_{\mathrm{N}} - 1. \tag{30}$$

This holds for any number of logical observables, including cases in which both predictions are incorrect. Taking conditional expectations and using Eq. (29) gives the backend lower bound

$$\mathbb{E}[e_{\mathrm{B}} \mid \mathcal{A}] \geq 1 - \mathbb{E}[e_{\mathrm{N}} \mid \mathcal{A}] \geq 1 - u_A. \tag{31}$$

Assumption 1(ii) implies $u_A < 1/2$, placing the neural upper bound in Eq. (29) strictly below the backend lower bound in Eq. (31). The difference of the actual conditional risks satisfies

$$\mathbb{E}[e_{\mathrm{N}} - e_{\mathrm{B}} \mid \mathcal{A}] \leq 2u_A - 1 < 0. \tag{32}$$

On $\mathcal{C} \setminus \mathcal{A}$, the predictions coincide and hence $e_{\mathrm{N}} = e_{\mathrm{B}}$.

Deferred changes bound the remaining contribution. On the defer branch, the guided and unassisted backends have equal error indicators whenever their logical predictions agree. In all cases,

$$e_{\mathrm{G}} - e_{\mathrm{B}} \leq \mathbf{1}[\hat{\boldsymbol{y}}_{\mathrm{G}}^{\mathrm{logical}}(\boldsymbol{s}) \neq \hat{\boldsymbol{y}}_{\mathrm{B}}^{\mathrm{logical}}(\boldsymbol{s})]. \tag{33}$$

The actual overall risks are $R_{\mathrm{CoD}} = \mathbb{E}[\mathbf{1}_{\mathcal{C}} e_{\mathrm{N}} + \mathbf{1}_{\mathcal{C}^c} e_{\mathrm{G}}]$ and $R_{\mathrm{B}} = \mathbb{E}[e_{\mathrm{B}}]$. Splitting their difference over the two branches gives

$$\begin{aligned} R_{\mathrm{CoD}} - R_{\mathrm{B}} &= \mathbb{E}[\mathbf{1}_{\mathcal{C}}(e_{\mathrm{N}} - e_{\mathrm{B}})] + \mathbb{E}[\mathbf{1}_{\mathcal{C}^c}(e_{\mathrm{G}} - e_{\mathrm{B}})] \\ &= \alpha\, \mathbb{E}[e_{\mathrm{N}} - e_{\mathrm{B}} \mid \mathcal{A}] + \mathbb{E}[\mathbf{1}_{\mathcal{C}^c}(e_{\mathrm{G}} - e_{\mathrm{B}})] \\ &\leq -\alpha(1 - 2u_A) + \zeta. \end{aligned} \tag{34}$$

Since $\alpha > 0$, Assumption 1(ii) is equivalent to

$$\alpha(1 - 2u_A) > \zeta. \tag{35}$$

This makes the right-hand side of Eq. (34) strictly negative, proving Eq. (16). The proof uses the logical outputs of both backends and therefore also includes outputs from nonconverged decoding runs. □

Averaging over committed shots. For $c = \Pr(\mathcal{C}) > 0$, the same pointwise bound gives the logical risk of the entire commit branch:

$$\Pr[\boldsymbol{L}\boldsymbol{y}^{\mathrm{fault}} \neq \boldsymbol{L}\hat{\boldsymbol{y}}^{\mathrm{fault}} \mid \mathcal{C}] \leq \mathbb{E}\left[\min\left\{1, \frac{S_{\boldsymbol{q}}(\boldsymbol{s}) + \varepsilon(\boldsymbol{s})}{d_{\log}}\right\} \middle| \mathcal{C}\right]. \tag{36}$$

An alternative sufficient condition can therefore use posterior accuracy averaged over all committed shots. If $\mathbb{E}[\varepsilon(\boldsymbol{s}) \mid \mathcal{C}] \leq \varepsilon_C$, define

$$u_C = \min\left\{1, \frac{\mathbb{E}[S_{\boldsymbol{q}}(\boldsymbol{s}) \mid \mathcal{C}] + \varepsilon_C}{d_{\log}}\right\}. \tag{37}$$

Then $\mathbb{E}[e_{\mathrm{N}} \mid \mathcal{C}] \leq u_C$. Since $\mathcal{A} \subseteq \mathcal{C}$, the decomposition above gives

$$R_{\mathrm{CoD}} - R_{\mathrm{B}} \leq 2\mathbb{E}[\mathbf{1}_{\mathcal{A}} e_{\mathrm{N}}] - \alpha + \zeta \leq 2c\, u_C - \alpha + \zeta. \tag{38}$$

Thus $\alpha > 2c\, u_C + \zeta$ also guarantees a strict improvement. These conditions concern the learned probabilities; the acceptance criterion remains the zero-residual test. The distance is defined on the full DEM and the measured observables. Any positive lower bound $0 < d_0 \leq d_{\log}$ may replace $d_{\log}$ in the risk bounds. If $\alpha = 0$, committed decisions coincide with the unassisted backend and $R_{\mathrm{CoD}} - R_{\mathrm{B}} \leq \zeta$ directly.

B.3 Training Loss Design

Reweighted fault loss. The final-round loss $\mathcal{L}_{\text{fault}}^{(R)}$ supervises individual fault labels $\boldsymbol{y}^{\text{fault}}$. Fired mechanisms are sparse, so we use a fixed positive-class weight w in binary cross-entropy to increase their contribution. The weight is the ratio of the expected number of non-fired to fired mechanisms under the training priors, clipped at a maximum. Because commitment and backend guidance use q_j as an estimate of $p_j(\boldsymbol{s})$, we correct the logit shift introduced by weighting. For $0 < p_j(\boldsymbol{s}) < 1$, minimising expected weighted binary cross-entropy yields $\operatorname{logit} p_j(\boldsymbol{s}) + \log w$ as the training logit. We therefore store $\log w$ as a constant offset and train using logits $\operatorname{logit} q_j + \log w$, while decoding uses q_j. This preserves the posterior target at the population optimum, with endpoint probabilities understood as limits.

Auxiliary logical loss. An auxiliary head pools the final mechanism states, weighted by $\boldsymbol{q}$, into each observable and predicts $\boldsymbol{y}^{\text{logical}}$. Its cross-entropy $\mathcal{L}_{\text{obs}}$ provides a direct logical training signal alongside the individual fault labels. The head's prediction is not used for decoding.

Deep supervision. For each intermediate round $t \in \mathcal{R}$, $\mathcal{L}_{\text{fault}}^{(t)}$ applies the same corrected fault loss using the shared readout on $\boldsymbol{h}_j^{(t)}$. These terms give intermediate states a direct target for posterior estimation before the final round.

C Experimental Setup and Evaluation Protocol

The circuit-level and code-capacity experiments in Section 4.1 use distinct noise models and separately trained neural predictors. Both compare decoders on frozen test populations, with the metrics and statistical conventions shared below.

C.1 Code Instances, Circuits, and Detector Error Models

Table 3: Code instances, memory experiments and the backends each is evaluated with. T is the number of syndrome-extraction rounds; D and m are the numbers of detectors and error mechanisms of the instance's detector error model.

Family	Instance	$[[n, k, d]]$	T	D	m	Basis	Backends
Bivariate bicycle	BB72	$[[72, 12, 6]]$	6	252	2232	Z	BP-OSD, Relay-BP
	BB108	$[[108, 8, 10]]$	10	594	5508	Z	BP-OSD, Relay-BP
	BB144	$[[144, 12, 12]]$	12	936	8784	Z	BP-OSD, Relay-BP
Quantum Tanner	RQT60	$[[60, 2, 6]]$	6	210	1140	X	BP-OSD, Relay-BP
	RQT108	$[[108, 18, 4]]$	4	240	2004	X	BP-OSD, Relay-BP
Surface	$d = 5$	$[[25, 1, 5]]$	5	120	1677	Z	MWPM, Relay-BP
	$d = 7$	$[[49, 1, 7]]$	7	336	5471	Z	MWPM, Relay-BP
	$d = 9$	$[[81, 1, 9]]$	9	720	12705	Z	MWPM, Relay-BP
Colour	$d = 5$	$[[19, 1, 5]]$	5	54	339	Z	BP-OSD, Relay-BP
	$d = 11$	$[[91, 1, 11]]$	11	540	4238	Z	BP-OSD, Relay-BP

The bivariate bicycle check matrices are constructed from $\boldsymbol{A} = \xi^3 + \eta + \eta^2$ and $\boldsymbol{B} = \eta^3 + \xi + \xi^2$ in commuting cyclic-shift variables ξ, η, with cyclic orders $(6, 6)$, $(9, 6)$ and $(12, 6)$ for BB72, BB108 and BB144, respectively, following Bravyi et al. [7]. The quantum Tanner parity-check matrices are taken from the public PropHunt benchmark artifact [46], using the $[[60, 2, 6]]$ and $[[108, 18, 4]]$ instances. We freeze these matrices and record the source revision and file hashes in the accompanying repository. The triangular 6.6.6 colour-code matrices are constructed from the open-boundary plaquette incidence pattern and supplied to qLDPC as CSS checks.

BB and colour circuits use the edge-colouring syndrome-extraction schedule in `qLDPC` version 0.2.9. Quantum Tanner circuits use the same package's edge-colouring schedule without inter-layer `TICK`

instructions. Surface circuits use Stim's generated rotated Z-memory schedule [17]. All instances use $T = d$ rounds and a noisy final readout round. Quantum Tanner codes store a logical X basis state; all other instances store a logical Z basis state (Tab. 3).

Noise and physical error rates. All circuits use SI1000 noise [19]. The LER curves in Fig. 2 and Fig. 5 use six to nine equally spaced values of p per instance, spanning the region below and around each instance's pseudo-threshold. Tab. 1 instead evaluates all ten instances at the common rate $p = 0.002$. For every p, the circuit, DEM and priors $\boldsymbol{\pi}$ are rebuilt; all decoders and pre-decoders read the same DEM.

The detector count D and mechanism count m in Tab. 3 are taken from the fixed circuit DEMs generated at $p = 0.002$, without splitting error mechanisms into smaller components. We merge mechanisms that trigger the same detectors and flip the same logical observables, and count each merged group once in m. Each merged group is assigned the probability that an odd number of its constituent mechanisms occur, since their effects cancel in pairs.

C.2 Code-Capacity Setting and Representation

The code-capacity experiments use perfect syndrome measurements and independent categorical Pauli errors across data qubits.

Specifically, each data qubit has $P(X) = 0.34p$, $P(Y) = 0.32p$, $P(Z) = 0.34p$, and $P(I) = 1 - p$, independently across qubits. Every method is evaluated at $p \in \{0.02, 0.03, \ldots, 0.10\}$ on BB72 ($d = 6$) and BB144 ($d = 12$).

This setting scores both X and Z logical components, so $\boldsymbol{y}^{\text{logical}}, \hat{\boldsymbol{y}}^{\text{logical}} \in \mathbb{F}_2^{2k}$ and $\boldsymbol{L} \in \mathbb{F}_2^{2k \times m}$. Accordingly, the logical-incidence entries in Eq. (4) extend through $L_{2k,j}$, giving $\boldsymbol{x}_j^{\text{m}} \in \mathbb{R}^{2k+2}$ and $\boldsymbol{E}_{\text{m}}, \boldsymbol{U}_{\text{m}} \in \mathbb{R}^{d_{\text{h}} \times (2k+2)}$. The auxiliary logical head predicts $2k$ components; detector features remain four-dimensional and node states retain width d_{h}. Here m denotes the number of binary fault components in the code-capacity representation; k still denotes the number of encoded logical qubits. Independence holds across qubits; the two binary Pauli components on one qubit can be correlated. The independent-mechanism product in Eq. (21) describes the circuit DEM, not this categorical Pauli channel.

C.3 Neural Implementation and Training

Circuit-level models. All circuit-level instances use AdamW with learning rate 3×10^{-4}, weight decay 10^{-4}, effective batch size 256, gradient-norm clipping at 1.0 and bfloat16 mixed precision. Microbatch sizes range from 16 to 256 according to memory requirements, with gradient accumulation maintaining the effective batch size. Each model is trained on approximately 10^7 shots sampled online. The shared network uses hidden width 96, $R = 10$ recurrent rounds, $K = 2$ active subgraphs, sum–max aggregation and prior-logit feature scale $\alpha_\pi = 0.1$. Deep supervision is applied at rounds 4 and 7; the loss coefficients are $\lambda_{\text{obs}} = \lambda_{\text{deep}} = 0.3$, with positive-class weight capped at $w_{\max} = 100$ and no focal-loss term.

For max aggregation, target nodes are grouped into degree ranges with power-of-two upper bounds. Neighbour states are padded to each bucket's width, padding entries are masked with $-\infty$, and a dense reduction computes the coordinatewise maxima. The bucket layout is reused across recurrent rounds within each forward pass.

Code-capacity models. Both BB72 and BB144 use roughly three million training samples per learned model. Training physical error rates are sampled uniformly from $[0, 0.15)$ for BB72 and $[0, 0.18)$ for BB144, with independent categorical Pauli errors in the proportions $X/Y/Z = 0.34/0.32/0.34$ and perfect syndrome measurements. These models are trained separately from the circuit-level models.

We use the original Astra model and follow its official implementation [32], with the training-data budgets described above. For BB72, model selection uses validation performance; for BB144, training ends at the fixed sample budget. The trained models are fixed across all test error rates. Tab. 2

concerns the common training-data scale. Appendix E.2 separately extends one BB72 Astra training trajectory to thirty million samples.

C.4 Baseline Decoders and Handoff Configurations

Each circuit-level instance uses the two backends listed in Tab. 3. BP-OSD runs min-sum BP with order-0 OSD on bivariate bicycle and colour codes, and product-sum BP with combination-sweep OSD of order 60 on quantum Tanner codes. Within each comparison, the pure and assisted BP-OSD decoders use the same backend configuration. Relay-BP uses the authors' implementation and published parameters, without an OSD fallback [34]. MWPM uses PyMatching on the decomposed surface-code DEM [25].

CoD learns circuit-level pre-decoding from approximately 10^7 shots per model, compared with the 10^8–10^9 shots used by AlphaQubit and the decoder of Blue et al. [2, 3]. The direct neural comparison tests data efficiency in Astra's published code-capacity setting using its official implementation [32], with roughly three million training samples per model and a BB72 Astra sweep extending to thirty million (Fig. 6).

CoD uses one candidate at threshold one half and the same trained model at every p of an instance. Prior blending for BP-OSD and MWPM uses $\beta = 0.25$; Relay-BP uses the conditioned search of Section 3.3.

C.5 Evaluation Metrics

Logical failure per complete memory experiment. Following the whole-experiment convention of Blue et al. [3], let $\boldsymbol{y}_i^{\text{logical}}, \hat{\boldsymbol{y}}_i^{\text{logical}} \in \mathbb{F}_2^k$ denote the true and predicted vectors of measured logical observables on shot i, and let N be the number of evaluated shots. We use

$$\widehat{P}_{\text{fail}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[\hat{\boldsymbol{y}}_i^{\text{logical}} \neq \boldsymbol{y}_i^{\text{logical}}]. \tag{39}$$

Thus, multiple incorrect observables on one shot count as one failure. No division by k or the number of syndrome-extraction rounds is applied, and no fidelity-decay inversion is used. All frozen shots are retained, including nonconverged backend predictions; convergence is a separate diagnostic.

The code-capacity LER (↓) counts shots with any incorrect logical component in the raw prediction. Nonconverged outputs are retained; syndrome inconsistency is recorded separately. Tab. 2 shows LER in percent.

Backend call rate and OSD call reduction. For a trained model ν, let $c_{i\nu}$ indicate whether the pre-decoder commits shot i, and $o_{i\nu}$ indicate whether BP-OSD invokes OSD on that shot. Backend call rate is

$$B_\nu = \frac{1}{N} \sum_{i=1}^{N} (1 - c_{i\nu}). \tag{40}$$

It counts deferred shots out of all test shots, equals one minus the commit fraction, and is shared by both assisted backends for a fixed neural model. Pure backends have $B_0 = 1$. Relay-BP counts each routed shot once, irrespective of internal search branches.

For BP-OSD, define the auxiliary absolute OSD call rate $O_\nu = N^{-1} \sum_{i=1}^{N} o_{i\nu}$ and let O_0 be its measured value for pure BP-OSD on the identical shots with the same backend configuration. The reported OSD call reduction (percent) is

$$R_\nu^{\text{OSD}} = 100 \left(1 - \frac{O_\nu}{O_0}\right). \tag{41}$$

Here $O_\nu \leq B_\nu$; both rates use all shots as the denominator, not only deferred shots. This relative reduction is negative when assistance increases OSD calls. OSD call reduction is N/A for MWPM and Relay-BP, and undefined when $O_0 = 0$. Backend call rate and OSD call reduction measure backend use.

C.6 Frozen Evaluation, Statistical Reporting, and Provenance

Paired evaluation and uncertainty. Methods within each row share frozen detectors, logical truth, ordered shot IDs, inclusion ranges and backend configurations. The main-table populations contain 50,000 shots per instance except for surface $d = 5, 7, 9$, which use 200,000, 250,000 and 950,000 shots, respectively. Pure backends report point estimates. We compute each trained model's metrics, including OSD call reduction, before the mean and sample SD [24]. The LER change column is $100(\overline{P}_{\text{fail}}/P_{\text{fail},0} - 1)$, computed before rounding, with downward/upward arrows for decreases/increases. Tab. 1 scales LER by 10^{-2} and rounds means and SDs to three decimal places; a displayed SD of 0.000 reflects rounding. The sample SD describes variation across trained models. Finite-shot uncertainty also matters at low LER, including when no errors are observed.

Code-capacity methods share 100,000 frozen shots per code and physical error rate. Fig. 6 follows a single BB72 Astra training trajectory on these populations.

Ablation evaluation. The subgraph-scope and aggregation comparisons in Appendix F evaluate models on identical frozen test shots within each comparison. The BB72 subgraph comparison uses 50,000, 200,000 and 50,000 shots at $p = 0.002$, 0.0025 and 0.004, respectively; the aggregation comparison uses the same 200,000-shot population at $p = 0.0025$.

The posterior-injection sweep uses the colour code at $d = 11$ and $p = 0.002$, with 50,000 frozen shots and the checkpoint at step 12,000 throughout. Input, checkpoint and source hashes accompany the results. Architecture ablations report individual evaluated configurations; timing repeats measure execution variability.

Astra training-data sweep. Training uses 30 recurrent rounds and inference uses 200, with the original test-batch grouping of 128 retained. Evaluation occurs at exact three-million-sample intervals, without model selection on test performance.

D Additional Circuit-Level Results

The circuit-level evaluation in Section 4.2 spans ten instances. Fig. 5 completes the error-rate curves, and Tab. 4 records the absolute rates underlying the main table's OSD call reductions.

D.1 Performance on the Remaining Code Instances

The remaining instances are BB108 ($d = 10$), BB144 ($d = 12$), RQT60 ($d = 6$), the surface code at $d = 5, 7$, and the colour code at $d = 5$. Fig. 5 compares assisted and unassisted decoding across each instance's physical error-rate grid under SI1000 noise.

The gains depend on the code and backend. The BP-OSD curves separate visibly on BB108 and BB144, whereas the Relay-BP curves on BB144 remain close. Tab. 1 gives the paired comparisons; Appendix F.3 and Appendix F.4 examine the commit and defer contributions on those populations.

D.2 Supporting Backend-Use Measurements

Tab. 4 records the absolute OSD rates used to compute Tab. 1; these auxiliary quantities are distinct from OSD call reduction. The rates use all shots and the same BP-OSD settings, following Appendix C.5. All pure backends have a 100% backend call rate. OSD is not applicable to MWPM or Relay-BP.

E Code-Capacity Comparison and Neural Inference Cost

Tab. 2 reports the BB72 and BB144 code-capacity LERs across $p = 0.02$–0.10, with methods in rows and the neural inference FLOP ratio in the bottom row. Fig. 4 pairs these LERs with neural computation in Section 4.3. The noise model, training data and statistical scope are specified in Appendix C.2, Appendix C.3 and Appendix C.6.

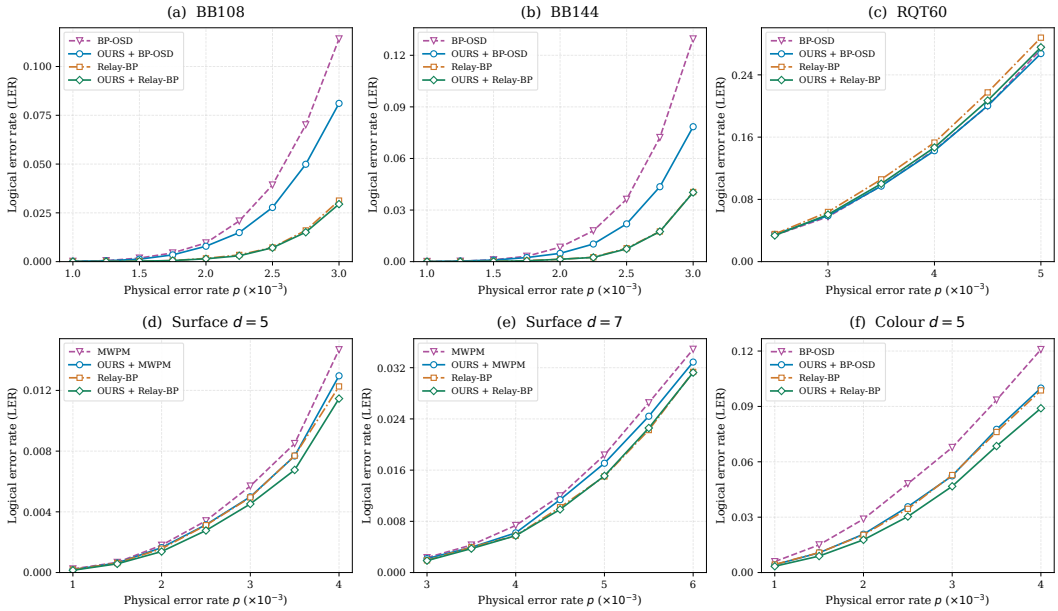


Figure 5: Logical error rate (↓) versus physical error rate on the six instances not shown in Fig. 2: (a) BB108, (b) BB144, (c) RQT60, (d)–(e) the surface code at $d = 5$ and $d = 7$, and (f) the colour code at $d = 5$, under SI1000 circuit-level noise. Each panel compares the two backends of the instance with and without CoD in front, evaluated on the same shots.

Table 4: Supporting absolute rates (percent) and test-set sizes at $p = 0.002$ for Tab. 1. Assisted rates are mean ± sample SD.

Code	Shots	Pure BP-OSD OSD call rate	CoD Backend call rate	CoD+BP-OSD OSD call rate
BB72	50,000	3.734	7.261±0.186	1.160±0.075
BB108	50,000	5.794	14.978±0.376	2.323±0.061
BB144	50,000	9.154	22.339±1.796	2.544±0.181
Surface $d = 5$	200,000	N/A	14.285±1.600	N/A
Surface $d = 7$	250,000	N/A	44.396±3.783	N/A
Surface $d = 9$	950,000	N/A	34.849±0.819	N/A
RQT60	50,000	22.300	17.093±2.254	6.675±0.443
RQT108	50,000	32.570	27.081±1.640	10.507±0.451
Colour $d = 5$	50,000	8.364	7.051±0.858	2.920±0.153
Colour $d = 11$	50,000	48.244	38.966±0.638	22.621±0.858

E.1 Neural Inference FLOP Accounting

For the code-capacity comparison in Fig. 4, neural FLOPs are analytical forward-pass counts with one multiply-accumulate counted as two operations. We count affine weight products and GRU gate matrices across the executed inference iterations. For CoD, the count uses the mean active graph sizes over all 100,000 test shots at each p and includes the auxiliary heads executed by the evaluation implementation. Astra follows the official implementation throughout, and its counted neural cost is constant across p for each code. Both assisted backends share the same neural-forward cost.

These counts exclude bias additions, activations, normalisation, graph construction, indexing, aggregation, logical pooling, consistency checks and classical backend operations. The comparison therefore measures the neural matrix-computation cost of inference.

On BB72, CoD requires 0.34–0.45 neural GFLOPs per shot, compared with 18.87 for Astra, a 42–56× reduction. On BB144, the corresponding costs are 0.79–0.90 versus 37.74 GFLOPs, a 42–48× reduction at matched physical error rates.

E.2 Sensitivity to Astra's Training-Data Budget

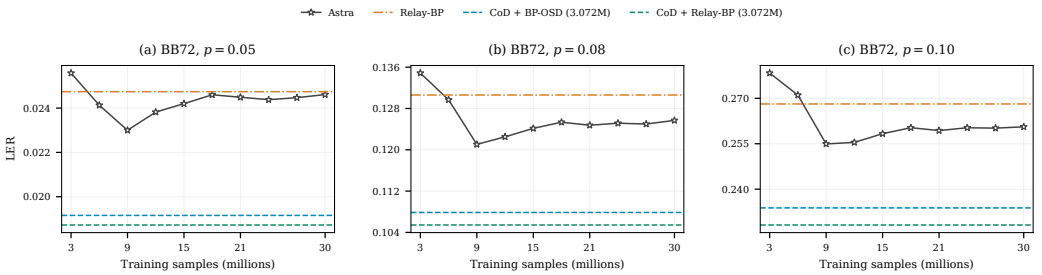


Figure 6: Astra on BB72 ($d = 6$) under code-capacity noise as a function of training samples, from three to thirty million, at $p \in \{0.05, 0.08, 0.10\}$. Horizontal lines show pure Relay-BP and the mean LERs of the two assisted backends from Tab. 2.

Fig. 6 evaluates one continuous BB72 Astra training trajectory from three million to thirty million samples using the published architecture and the evaluation protocol in Appendix C.6. From three to nine million samples, LER decreases by 10.1%, 10.2% and 8.4% at the three respective error rates, averaging about 9.6%. The lowest observed LER occurs at nine million samples at all three rates, about three times the training budget of CoD. With further training under the schedule used for this sweep, LER plateaus slightly above these minima. The later LERs lie at or slightly below pure Relay-BP and above both CoD+BP-OSD and CoD+Relay-BP trained on roughly three million samples (Tab. 2).

Extending Astra training to thirty million samples, roughly $10\times$ the training data used by CoD, does not close the accuracy gap. Both assisted decoders retain lower mean LERs than this Astra trajectory at all three physical error rates, together with the neural inference costs detailed in Appendix E.1.

F Mechanism Analysis and Ablation Studies

The analysis in Section 4.4 follows the decoding pipeline. Active-subgraph and aggregation comparisons assess the predictor in Section 3.1; routing diagnostics examine the commit rule in Section 3.2; deferred-shot comparisons and the posterior-injection sweep assess the handoff in Section 3.3. Evaluation populations and statistical conventions are given in Appendix C.6.

F.1 Active-Subgraph Scope

We compare full-graph message passing with K-hop active subgraphs for $K \in \{1, 2, 3\}$ on BB72 ($d = 6$) under six-round SI1000 circuit-level memory noise at $p \in \{0.002, 0.0025, 0.004\}$. The comparison fixes the dual-GRU architecture and sum–max aggregation. BP-OSD uses at most 300 BP iterations, OSD-0, and posterior-injection strength $\beta = 0.25$.

Tab. 5 shows that local computation improves both accuracy and efficiency at low noise. At $p = 0.002$ and $p = 0.0025$, $K = 2$ lowers LER from 2.3000% to 2.2460% and from 5.7145% to 5.6360%, respectively, giving the lowest LER among the tested scopes at both rates. It also accelerates training by $5.83\times$ and evaluation by 3.76–$5.14\times$ relative to the full graph. The $K = 1$ model provides the largest evaluation speedup, but has higher LER and lower OSD call reduction than $K = 2$ at all three tested noise levels. Increasing the radius from $K = 2$ to $K = 3$ improves OSD call reduction and lowers backend call rate at every tested noise level, and gives the lowest measured LER at $p = 0.004$. The $K = 2$ results directly support the active-subgraph design: concentrating message passing around fired detectors reduces computation and improves low-noise accuracy.

F.2 Message Aggregation

We compare sum-only, max-only, and concatenated sum–max aggregation on BB72 ($d = 6$) at $p = 0.0025$ under six-round SI1000 circuit-level memory noise. The comparison fixes the $K = 2$

Table 5: Subgraph scope ablation on BB72 ($d = 6$) at $p \in \{0.002, 0.0025, 0.004\}$. OSD call reduction uses the matched pure BP-OSD baseline. Speedups are relative to the full-graph implementation. Timing protocols are given in Appendix C.6. Bold denotes the best measured metric per noise setting.

Subgraph	p	LER (%) ↓	OSD call reduction (%) ↑	Backend call rate (%) ↓	Train speedup ↑	Eval speedup ↑
Full graph	0.0020	2.3000	67.7022	8.2580	1.00×	1.00×
	0.0025	5.7145	60.5214	17.6360		1.00×
	0.0040	32.8600	37.0456	63.1260		1.00×
$K = 1$	0.0020	2.3580	62.3460	9.9360	**11.81**×	**13.39**×
	0.0025	5.7280	56.4503	19.4690		**12.01**×
	0.0040	33.0760	35.5007	63.4360		**6.38**×
$K = 2$	0.0020	**2.2460**	66.6310	7.1240	5.83×	5.14×
	0.0025	**5.6360**	60.7354	15.9975		4.92×
	0.0040	32.9840	36.1823	61.5260		3.76×
$K = 3$	0.0020	2.3360	**68.5592**	**6.6720**	5.54×	5.14×
	0.0025	5.7570	**62.5655**	**14.6125**		4.98×
	0.0040	**32.7440**	**40.0991**	**57.4120**		3.82×

Table 6: Aggregation ablation on BB72 ($d = 6$) at $p = 0.0025$: K-hop active subgraphs with $K = 2$. OSD call reduction uses the matched pure BP-OSD baseline. Bold denotes the best measured value.

Aggregation	LER (%) ↓	Backend call rate (%) ↓	OSD call reduction (%) ↑
Sum only	6.0420	19.9790	52.6043
Max only	6.0550	27.3065	51.0389
Sum + max	**5.6360**	**15.9975**	**60.7354**

active subgraph and dual-GRU architecture, with the same BP-OSD configuration as in the subgraph-scope comparison.

Tab. 6 shows that sum–max aggregation achieves the lowest measured LER and backend call rate, together with the highest OSD call reduction. Relative to sum-only and max-only aggregation, it reduces LER by approximately 6.7% and 6.9%, respectively; its OSD call reduction relative to pure BP-OSD is 60.74%. These results support combining sum and max aggregation in the final model for this setting.

F.3 Commit Routing and Reliability

Tab. 7 measures the routing and reliability of neural candidates on the same frozen populations as Tab. 1. These diagnostics separate the commit behaviour from the guided-backend results in Appendix F.4.

Definitions and direction. For each trained model evaluated on N shots, define the sample-index sets $\mathcal{I}_{\text{commit}} = \{i \in \{1, \dots, N\} : \boldsymbol{r}_i = \mathbf{0}\}$ and $\mathcal{I}_{\text{defer}} = \{1, \dots, N\} \setminus \mathcal{I}_{\text{commit}}$. These are the empirical subsets corresponding to the events $\mathcal{C}$ and $\mathcal{C}^c$ in Section 3.2; D continues to denote the number of detectors. Let $\mathcal{I}_{\text{err}} = \{i \in \{1, \dots, N\} : \boldsymbol{L}\hat{\boldsymbol{y}}_i^{\text{fault}} \neq \boldsymbol{y}_i^{\text{logical}}\}$ contain the indices of shots on which the original neural hard candidate has at least one incorrect logical observable. Backend call rate (↓) remains $|\mathcal{I}_{\text{defer}}|/N$, exactly as in Tab. 1. LER on committed shots (↓) is $|\mathcal{I}_{\text{err}} \cap \mathcal{I}_{\text{commit}}|/|\mathcal{I}_{\text{commit}}|$. Neural logical-error capture rate (↑) is $|\mathcal{I}_{\text{err}} \cap \mathcal{I}_{\text{defer}}|/|\mathcal{I}_{\text{err}}|$: the fraction of neural candidate logical errors routed to a backend. The preferred directions must be read jointly, since deferring every shot would capture every candidate error while eliminating no backend calls.

Neural candidate mismatch on deferred shots is $|\mathcal{I}_{\text{err}} \cap \mathcal{I}_{\text{defer}}|/|\mathcal{I}_{\text{defer}}|$, measured before backend decoding. It describes the neural candidate rather than physical faults or the decoded backend output, and has no unconditional preferred direction: a higher value can reflect more selective routing, but

Table 7: Commit routing and reliability at $p = 0.002$ under SI1000 noise, for all ten instances in Tab. 1. Values are percentages, mean ± sample SD. Arrows indicate the preferred direction when read jointly. Deferred candidate mismatch is a diagnostic with no unconditional preferred direction. Definitions are in Appendix F.3.

Code	Backend call rate (%) ↓	LER on committed shots (%) ↓	Neural candidate mismatch on deferred shots (%)	Neural logical-error capture rate (%) ↑
BB72	7.2607±0.1856	0.8633±0.0251	67.1545±1.5351	85.8905±0.4186
BB108	14.9780±0.3764	0.0023±0.0024	58.6041±1.4890	99.9773±0.0227
BB144	22.3387±1.7965	0.0102±0.0042	58.8643±0.4574	99.9377±0.0318
Surface $d = 5$	14.2848±1.6004	0.0387±0.0047	13.2707±2.4341	98.1640±0.7561
Surface $d = 7$	44.3963±3.7832	0.0005±0.0004	10.1849±1.0854	99.9944±0.0050
Surface $d = 9$	34.8485±0.8193	0.0001±0.0001	7.4162±0.6071	99.9985±0.0026
RQT60	17.0927±2.2539	0.5779±0.0365	28.0536±2.5125	90.6936±2.3551
RQT108	27.0807±1.6404	6.4252±0.4076	76.9348±0.7387	81.5849±2.0484
Colour $d = 5$	7.0507±0.8576	0.8895±0.1142	40.6621±4.2300	77.4892±2.6992
Colour $d = 11$	38.9660±0.6380	0.0284±0.0035	24.8865±0.4606	99.8216±0.0233

can also result from worse neural predictions. A deferred candidate with matching logical observables is not necessarily syndrome-consistent.

Statistical scope. Statistics follow Appendix C.6 and use the existing predictions on the frozen shots. Percent LER is numerically identical to the main table's $\times 10^{-2}$ units; four decimal places expose small conditional rates.

Routing and reliability. Tab. 7 shows selective routing: across all ten instances, the mean fraction of neural logical errors deferred exceeds the mean fraction of shots deferred. Backend call rates of 7.05–44.40% coexist with neural logical-error capture rates of 77.49–99.999%. For surface $d = 5$, for example, the check defers 14.2848% of shots while capturing 98.1640% of neural candidate logical errors, with a committed LER of 0.0387%. This supports the mechanism's ability to concentrate candidate errors in the deferred branch while bypassing the backend on many shots.

Committed LER varies across instances, from 0.0001% to 6.4252% (Tab. 7), reflecting differences in the logical risk of the syndrome-consistent candidates. Proposition 1 gives a sufficient condition for logical-risk improvement in terms of posterior accuracy, uncertainty and branch disagreements.

F.4 Backend Guidance on Deferred Shots

In Tab. 8, all four LERs (↓) count shots with any incorrect logical observable after decoding and use the same deferred-set denominator $|\mathcal{I}_{\text{defer}}|$. The columns for B and Relay-BP restrict pure-backend predictions to that subset; the assisted columns use the corresponding guided predictions.

Tab. 8 isolates backend guidance on fixed deferred subsets. Guidance lowers the mean deferred LER for B on eight of ten instances and for Relay-BP on nine of ten, with increases in the remaining comparisons. The table reports every paired result and its sample SD; statistical interpretation follows Appendix C.6.

The BB144 and RQT60 results explain the two overall mean-LER increases in Tab. 1. On BB144 ($d = 12$, $p = 0.002$), unassisted Relay-BP already decodes the deferred subset with 0.5662% mean LER, compared with 3.6137% for unassisted BP-OSD. Neural guidance reduces the latter to 2.1211% but gives 0.5825% for Relay-BP. The same neural model therefore improves BP-OSD substantially while adding no accuracy gain to the stronger Relay-BP backend on this subset.

On RQT60 ($d = 6$, $p = 0.002$), pure BP-OSD invokes its OSD search on 22.30% of all shots (Tab. 4). Guidance leaves the deferred-shot mean LER nearly unchanged at 7.0663% versus 7.0240% without guidance, while the committed-shot LER is 0.5779% (Tab. 7). The remaining commitment errors receive no compensating accuracy gain from guidance. Here CoD reduces backend calls by 82.91% and OSD calls by 70.07%, with a 1.70% relative increase in overall mean LER. These cases show

Table 8: LER (%, ↓) on the same deferred shots at $p = 0.002$ under SI1000 noise. Each entry is mean ± sample SD. B is MWPM for surface codes and BP-OSD otherwise. Pure-backend predictions are restricted to the corresponding deferred set; these are conditional LERs, not the full-population baselines in Tab. 1.

Code	B	CoD+B	Relay-BP	CoD+Relay-BP
BB72	26.0875±0.7268	20.0860±0.3677	20.3930±0.5567	16.2920±0.4138
BB108	6.1711±0.1246	5.2551±0.1143	1.0820±0.0273	0.9571±0.0376
BB144	3.6137±0.2710	2.1211±0.1054	0.5662±0.0416	0.5825±0.0752
Surface $d = 5$	0.9936±0.0743	0.9168±0.0083	0.8756±0.0726	0.7392±0.0420
Surface $d = 7$	0.1077±0.0093	0.0966±0.0082	0.1053±0.0090	0.1017±0.0102
Surface $d = 9$	0.0296±0.0010	0.0230±0.0001	0.0859±0.0021	0.0746±0.0015
RQT60	7.0240±0.7651	7.0663±0.7737	7.2950±0.8291	7.1539±0.7866
RQT108	38.2031±0.8241	38.4048±0.4484	40.1822±0.6781	37.4609±0.8570
Colour $d = 5$	21.6942±1.4264	17.8337±1.2929	15.7496±0.8995	13.4293±0.7962
Colour $d = 11$	3.7199±0.0420	2.7677±0.0999	1.7795±0.0191	1.6892±0.0796

that accuracy gains depend on both reliable commitment and room to improve the backend's deferred decisions.

The branch diagnostics measure commitment reliability and compare guided and unassisted backends on the same deferred shots. The overall comparisons in Tab. 1 combine the commit and defer contributions.

F.5 Sensitivity to Posterior-Injection Strength

We vary the posterior-injection strength β on the colour code with $d = 11$, keeping the neural commit decision fixed. The $\beta = 0$ control retains neural commitment and removes posterior injection, isolating the additional benefit of guidance on deferred shots.

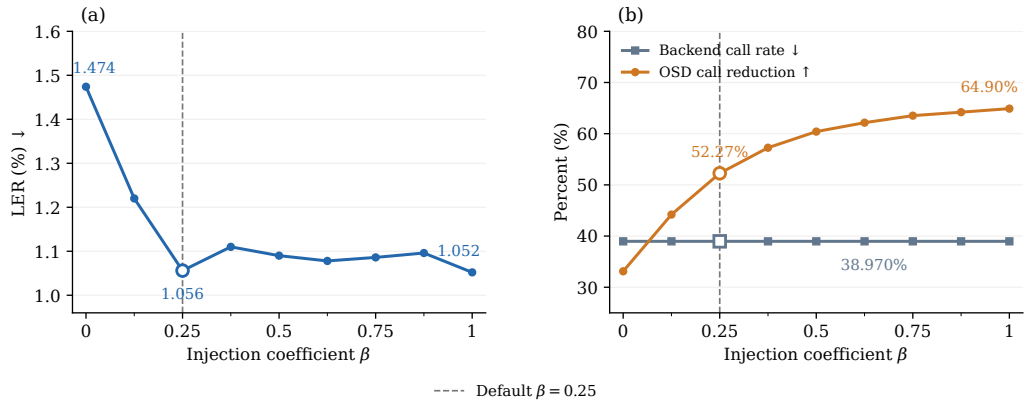


Figure 7: Sensitivity to posterior-injection strength on the colour code with $d = 11$ at $p = 0.002$. (a) LER. (b) Backend call rate and OSD call reduction relative to pure BP-OSD. The dashed line marks the default $\beta = 0.25$.

Fig. 7 shows that increasing β from 0 to 0.25 lowers LER from 1.474% to 1.056%, a relative reduction of 28.4%. Across $\beta \in [0.25, 1]$, LER remains between 1.052% and 1.110%, so the LER improvement persists across this range in the tested setting. Over this interval, OSD call reduction increases from 52.27% to 64.90%, whereas backend call rate stays at 38.970% because β does not affect the commit decision. These results support the default $\beta = 0.25$ while distinguishing the accuracy benefit of posterior injection from the unchanged routing decision.