TRAIN ON THE SYNDROME STREAM YOU WILL DECODE

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantum error correction (QEC) turns repeated syndrome measurements into correction decisions. Neural decoders have gained widespread attention because they can learn complex, hardware-specific noise correlations directly from data. Neural decoders are usually trained and judged on reset-bounded memory experiments, assuming that benchmark winners remain best during uninterrupted computation. However, we find that this ranking can reverse. On never-reset surface-code streams at $d = 5, 7, 9$, convolutional checkpoints that beat correlated matching by 12-17% on memory benchmarks fall behind it by 1.5-1.7$\times$ on interior decisions, while released graph and transformer decoders show the same failure. To identify the source of this reversal, we use controlled interventions, which trace it to preparation and terminal-readout boundaries rather than sequence length. Building on this diagnosis, we fine-tune on interior-window labels. This restores the neural lead across all three families and, with one-window lookahead, reduces convolutional per-commit error by 40-45% versus matching in simulation. The same lesson holds for lattice surgery: a better frame predictor need not yield a better conditional parity, whereas joint frame–readout supervision improves the combined decision, and the jointly trained system lowers conditional parity error by 9-15% versus joint matching. Together, these findings make existing neural decoders stream-ready with a single fine-tuning stage and pave the way for neural decoding in fault-tolerant quantum computation.

1 INTRODUCTION

Quantum error correction (QEC) is the foundation for fault-tolerant quantum computation (Dennis et al., 2002; Fowler et al., 2012). Deep quantum circuits require error rates far below physical-qubit levels, and QEC bridges this gap by encoding logical information redundantly across many physical qubits. A decoder then infers errors from measured syndromes and prescribes corrections. This has motivated extensive work on surface-code decoders, with minimum-weight perfect matching the most widely used (Higgott, 2022; Higgott & Gidney, 2025). Recent surface-code memory experiments using correlated matching have shown logical error decreasing with code size in the below-threshold regime (Google Quantum AI and Collaborators, 2025). Beyond memory experiments, encoded logical circuits have been executed on reconfigurable atom arrays (Bluvstein et al., 2024). Together, these advances move QEC toward large-scale fault tolerance, where larger codes and better decoding can further suppress logical errors during computation.

Recently, neural decoders have emerged as a data-driven alternative to matching. They learn correlations across qubits and measurement rounds that are difficult to encode in a matching model, and recent models beat correlated matching on memory benchmarks (Bausch et al., 2024; Yan et al., 2026a;b; Senior et al., 2025). However, these benchmarks score decoding by a final readout, whereas a fault-tolerant computation also needs decisions while it is still running. Moreover, memory-trained decoders learn from examples that begin with a prepared state and end with a final readout, whereas decisions during continuous operation are made in windows inside an ongoing syndrome stream, with neither boundary. This leaves an open question:

Does a decoder that wins a memory benchmark retain its lead on continuous-stream decisions?

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.

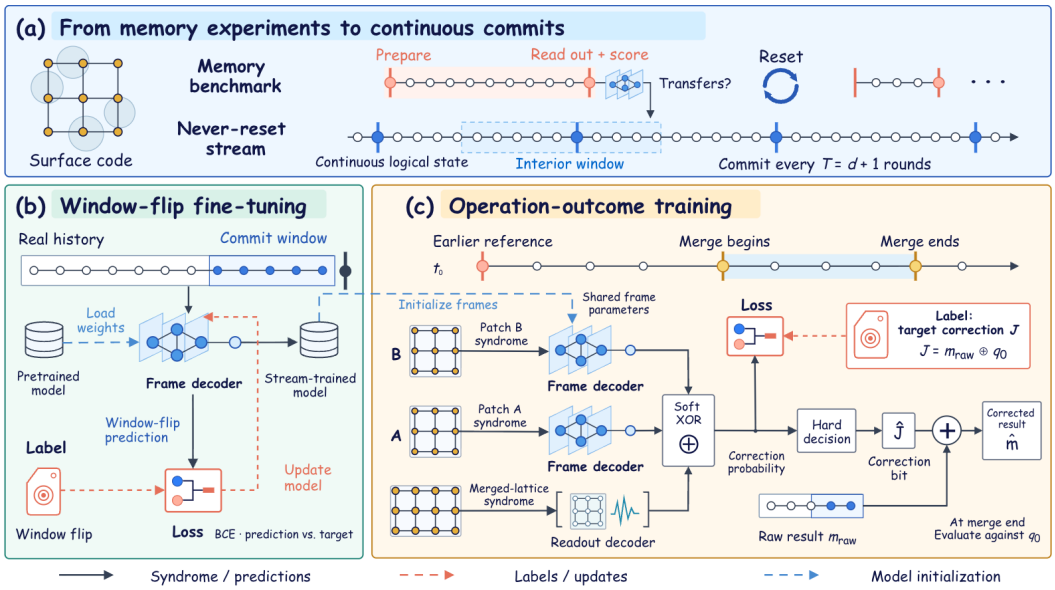


Figure 1: **Training for continuous commits and conditional parity.** (a) Reset-bounded memory experiments and a never-reset stream with commits every $T = d + 1$ rounds. (b) Window-flip fine-tuning from memory-trained weights, with reference taps supplying labels only. (c) Joint training of the shared frame network and merge-readout network on the conditional parity target J at $B = 1$, scored after T merge rounds relative to the pre-block reference q_0. Dark solid arrows denote syndrome data and predictions. Red dashed arrows denote labels or updates, and blue dashed arrows denote weight initialization.

Existing neural decoders are largely developed around reset-bounded memory experiments. Their training examples begin with state preparation and end with final readout, and their performance is reported in the same bounded setting. Thus, both training and evaluation are centered on memory experiments rather than continuous operation. We instead evaluate decoders on interior decisions from never-reset streams and fine-tune neural decoders on the same kind of interior windows. This provides a fairer comparison under continuous operation and, by training on the same interior-window task, improves neural decoding performance in that setting.

Concretely, we evaluate never-reset surface-code streams at code distances $d = 5, 7, 9$ (up to 161 physical qubits). On these streams, simulator-only reference measurements supply labels for interior windows but never enter the decoder inputs. We compare our convolutional decoder with released pretrained transformer (Yan et al., 2026b) and graph (Lange et al., 2025) decoders. To investigate the role of training, we vary one condition at a time in the convolutional family and evaluate interior-window fine-tuning across all three families. We further examine how frame predictions combine with readout corrections in a lattice-surgery measurement of two encoded qubits, and test composed window predictions on public device records (Google Quantum AI, 2023; 2024). Our experiments yield three findings.

(i) Memory-benchmark rankings reverse on never-reset streams. We establish an evaluation protocol for interior commits under a common syndrome cutoff. On these streams, memory-trained convolutional decoders that beat correlated matching by 12-17% on memory experiments fall behind it by 1.5-1.7×. Controlled interventions show how preparation and readout boundaries change this ordering, identifying a training mismatch relevant to continuous error correction.

(ii) Fine-tuning on interior windows restores the lead across three decoder families. We fine-tune memory-trained decoders on interior-window labels and restore the lead over correlated matching for convolutional and graph decoders at $d = 5, 7, 9$ and for the released transformer at $d = 7$. With one window of lookahead, our convolutional decoder reduces per-commit error by 40–45% relative to correlated matching in simulation, with the same buffer. This establishes a practical training stage for adapting existing decoders to decisions within an ongoing measurement stream.

(iii) Component rankings can reverse after frame–readout composition. For a lattice-surgery measurement of two encoded qubits, we jointly train the frame and readout networks on a conditional parity target defined relative to the noisy state before the scored block. Across all three matched training runs at $d = 5$, joint supervision lowers conditional error relative to separate supervision, despite higher component errors.

2 PRELIMINARIES

Surface code and syndrome. Surface codes protect logical information using local stabilizer measurements and extended logical operators (Dennis et al., 2002; Fowler et al., 2012). A rotated surface code of distance d encodes one logical qubit in d^2 data qubits, and the d^2-1 measurement qubits between them each measure a stabilizer, the parity of their neighbouring data qubits (Google Quantum AI, 2023). One round of syndrome extraction measures every stabilizer, about once per microsecond on a superconducting processor (Google Quantum AI and Collaborators, 2025). A detector compares a stabilizer with its value in the previous round; a detector compares successive stabilizer outcomes; a detection event occurs when they disagree. Errors fire detectors along their path, and a chain of errors across the lattice flips the logical observable, the parity of a line of data qubits. The decoder reads the detection events, one layer of $d^2 - 1$ bits per round, and decides whether the observable flipped. We simulate with Stim (Gidney, 2021) under circuit-level depolarizing noise at physical error rate p.

The memory experiment and its decoders. The processor prepares the data qubits in a known eigenstate of the observable, runs r rounds, measures every data qubit, and resets. One run of this kind is a shot, and its record is a block of detection events with one final readout layer whose label is whether the readout disagrees with the prepared state. Memory benchmarks report decoder performance as logical error per experiment. Minimum-weight perfect matching decodes that block on the detector error model of the circuit (Higgott, 2022; Higgott & Gidney, 2025), and correlated matching runs it twice, letting the errors found in one pass reweight the other. It is the decoder of the below-threshold experiments (Google Quantum AI and Collaborators, 2025) and the baseline throughout. A neural decoder is a classifier $f(s; \theta)$ that reads the block s and outputs the probability that the observable flipped, trained by cross-entropy on simulated samples (Bausch et al., 2024; Yan et al., 2026a). We compare several trained checkpoints of the same architecture.

Decoding a stream. A computation does not end after r rounds. Outcomes measured mid-circuit feed forward into the operations after them (Bluvstein et al., 2024). Known Pauli corrections can be tracked classically and used to interpret later measurement outcomes (Fowler et al., 2012). The decoder therefore processes an ongoing syndrome stream and supplies decisions at the required deadlines. Window decoders divide the stream into windows. They commit the corrections in a commit region and hold the rest in a buffer for the next window, and a buffer of d rounds keeps the full code distance for matching (Skoric et al., 2023). The buffer a commit is given is the number of future rounds made available before its decision. This lookahead contributes to the waiting time of an operation that consumes the decision.

3 PROBLEM FORMULATION

3.1 TWO EVALUATIONS

Memory experiments and continuous operation present decoders with different syndrome records. Figure 1 follows decoding from a memory benchmark to the decisions required by continuous operation. In panel (a), each memory experiment begins with preparation, ends with data-qubit readout and reset, and yields one scored outcome; performance is reported as logical error per experiment (Bausch et al., 2024; Yan et al., 2026a;b; Varbanov et al., 2025). The never-reset stream instead carries a logical state across decisions, with a commit every $T = d+1$ rounds. The question is whether a decoder trained on bounded memory runs can predict changes in a fixed logical observable within these interior windows.

Panel (b) shows our adaptation: a frame decoder initialized from memory-trained weights receives real predecessor history and the current commit window. A simulator-provided window-flip label supervises its prediction without entering its input. Panel (c) shows the operation-level decision.

Frame outputs for patches A and B, produced with shared parameters, combine with a merge-readout output through a soft XOR. The resulting correction probability is trained against $J = m_{\mathrm{raw}} \oplus q_0$. At the merge deadline, its hard decision $\hat{J}$ corrects the raw result to $\hat{m} = m_{\mathrm{raw}} \oplus \hat{J}$, which is evaluated against the earlier reference q_0.

We study an uninterrupted syndrome record with no preparation or final readout inside the scored window. The schedule uses $T = d + 1$ rounds per commit, motivated by lattice-surgery decoding deadlines (Skoric et al., 2023; Zhang et al., 2026b). Each target is the net flip of the circuit's fixed logical representative across that window. Every decoder is scored against this target using the same syndrome cutoff. XOR-composing B commits scores the increment over their combined block (Figure 10). A buffer of one window supplies T further rounds before the decision. The metric is fixed-representative increment error. Section 4.3 evaluates the composition of these increments with readout correction, using the noisy pre-block state as its reference.

The baseline throughout is correlated minimum-weight perfect matching (Google Quantum AI and Collaborators, 2025), correlated matching for short, implemented with the package of Higgott (2022) and decoding the same records. For interior commitment it takes a causal form with a bounded history band, and widening the tested history band leaves its decisions unchanged (Appendix C). Its decisions use the circuit's detector error model and the observed detection events. Belief propagation with localised statistics reads the same commits, ahead of correlated matching at distances 5 and 7 and behind it at 9, and the stream-fine-tuned decoder outperforms both at all three distances (Appendix C). The same appendix also compares the training boundaries, supervision targets, and evaluation targets of related decoding protocols.

3.2 The history available to a commit

The deployed network has L residual blocks, each with a width-3 temporal convolution at unit dilation. The final width-3 scatter convolution adds two rounds, so each output round depends on at most $R = 2L + 3$ input rounds. Pooling a T-round window can therefore use at most $R - 1$ preceding rounds. With $L = d$ and $T = d + 1$, this bound is $R - 1 = 2T$. Appendix A derives the support bound, verifies its attainment for the tested output scores, and proves equality between stateful streaming and replay with this history. These results fix the context used by the decoder and allow the same weights to be compared across input-boundary conditions.

3.3 Training on interior windows

The history bound specifies the context a commit can access. We test how training on that context affects interior-window predictions. At distance 5, training with no predecessor reaches the constant predictor's error, whereas one window of real predecessor history already gives a mean error below matching. Two and three windows do not improve on it (Appendix B). We use three windows for the stream fine-tuning. The boundary controls in Section 4.1 vary preparation, readout and predecessor context separately while holding the training budget and initialization fixed within the control family.

Stream fine-tuning. For no-buffer decoding, we fine-tune the memory-trained decoder in one stage on never-reset streams. Two choices define its data. First, every commit in a training stream is interior, with three windows of real predecessor behind it and no readout closing it. Second, the label is the net flip of the observable across the window, read on simulated streams from noiseless reference taps that enter the label and never the input (Appendix C). On a device run, whose interior carries no label, the decoder commits every window, the window decisions are composed by the parity rule, and the run's recorded readout supervises that composition, causally here and with a buffer in Zhang et al. (2026b) (Appendix E). The stage retains the architecture, starting weights, optimizer family, training noise level and loss, and runs 40,000 further steps at a lower constant learning rate and a smaller batch (Appendix G). The single-factor controls of Section 4.1 share a stitched-trained initialization. The buffered stage also starts there (Appendix B.2). A decoder built by others receives the same stage at its own training noise level (Section 4.2).

The streaming state. The streaming implementation carries the recent convolution inputs in queues containing $R-1$ feature tensors in total. For a window preceded by at least $R-1$ processed rounds, this implementation returns the same score as replay. Appendix A gives the state construction, proof and numerical verification. Appendix G describes the kernel that runs it.

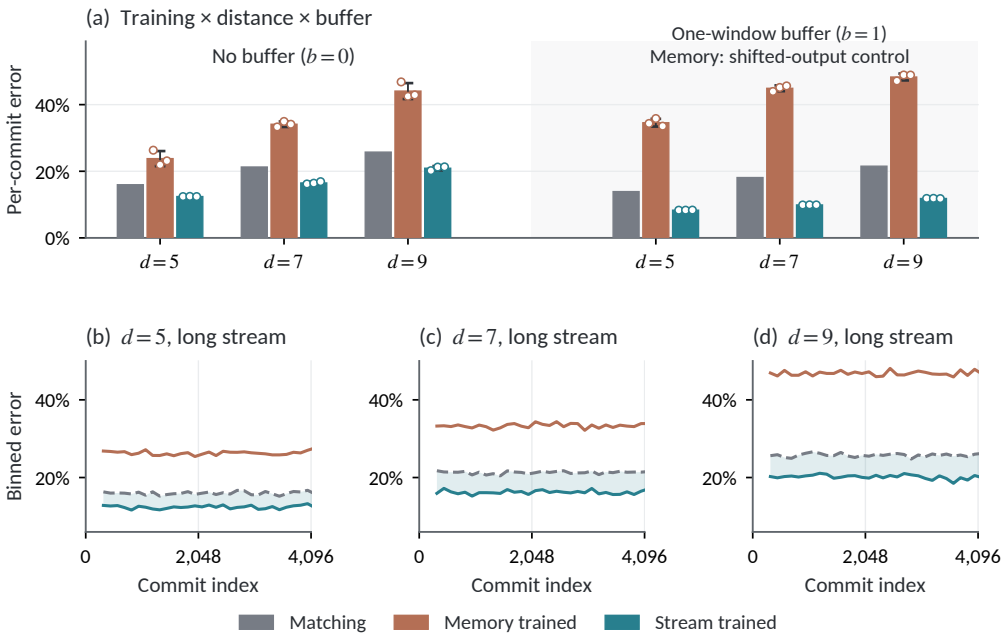


Figure 2: **Stream decoding across distance, buffer, and time.** (a) Per-commit error for memory-trained and stream-fine-tuned decoders against matching with the same buffer. Bars show three-run means ± sample SD. Open dots show individual runs. Buffered memory-weight bars are shifted-pooling controls (Appendix B.2). (b–d) No-buffer errors at $d = 5, 7, 9$ for one trained checkpoint per distance, averaged in 32 bins of 128 commits over 64 trajectories. Dashed curves mark matching. Shading shows its gap from stream fine-tuning. All panels use $p = 0.007$.

Table 1: **Decoder ranking across memory, causal commits and buffered commits.** Errors (%) at $p = 0.007$: per experiment on 200,000 isolated d-round memory shots, or per $T = d + 1$-round commit on 262,144 never-reset commits. The isolated and no-buffer memory-trained entries use identical weights. In the buffered block, Memory is a shifted-pooling control without buffer-target retraining. Fine-tuned starts from stitched-trained weights and trains on the buffered target (Appendix B.2). Neural entries show means ± SD over three training runs. **Bold** marks the lowest error within each protocol and distance.

	Isolated memory		Stream: no buffer			Stream: one-window buffer		
d	Memory	Matching	Memory	Matching	Fine-tuned	Memory	Matching	Fine-tuned
5	**2.46 ± 0.02**	2.78	23.80 ± 2.26	15.96	**12.42 ± 0.03**	34.57 ± 1.16	13.90	**8.33 ± 0.02**
7	**1.91 ± 0.04**	2.30	34.12 ± 0.83	21.28	**16.50 ± 0.35**	44.91 ± 0.86	18.12	**9.90 ± 0.02**
9	**1.51 ± 0.04**	1.79	44.07 ± 2.39	25.76	**20.92 ± 0.64**	48.29 ± 1.03	21.53	**11.83 ± 0.04**

4 Results

4.1 The main result on never-reset streams

The memory-benchmark ordering reverses on the stream. Table 1 compares the same memory-trained checkpoints across both protocols: all three runs at every distance beat matching on isolated memory experiments and lose to it on the stream, before any fine-tuning. At each of distances 5, 7 and 9 we score 4,096 commits on each of 64 never-reset trajectories, which carry no state preparation anywhere inside them, and every decoder is scored on the same commits (Appendix C). With one window of buffer, the construction the streaming literature deploys, our stream-fine-tuned decoder outperforms correlated matching by 40%, 45% and 45% at $d = 5$, 7 and 9 given the same buffer, and the memory-trained decoder under that same budget has a higher error than the constant predictor (Figure 2a, Table 5). Removing the buffer narrows the margin and leaves the ordering: the memory-trained decoder loses to causal correlated matching at every distance and the fine-tuning stage puts all nine training runs ahead of it. Across the same three runs and 64 trajectories, the

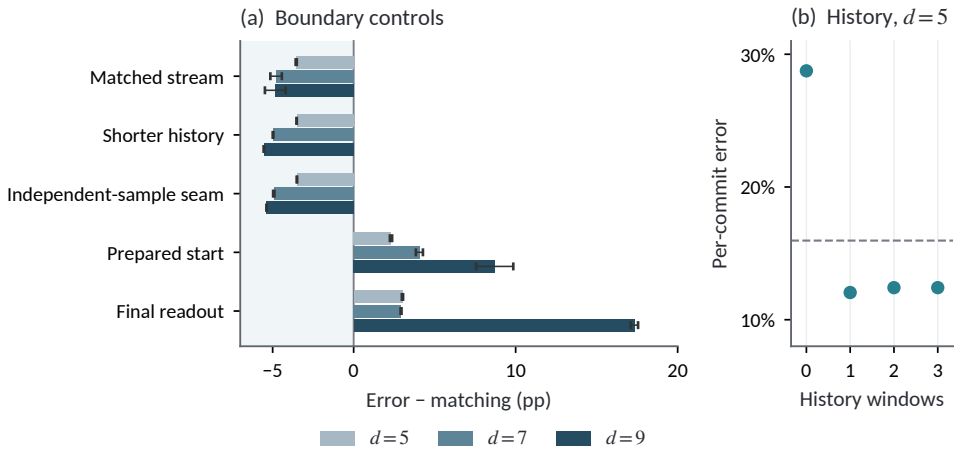


Figure 3: **Training-window boundaries and predecessor history.** (a) Per-commit error minus causal matching, in percentage points, for five training conditions at $d = 5, 7, 9$. Negative values favor the learned decoder. (b) Per-commit error versus training-history length at $d = 5$. The dashed line marks causal matching. Both studies use no buffer and $p = 0.007$, with means $\pm$ sample SD over three runs. The studies use separate training runs.

absolute mean error reduction grows with distance under both buffer budgets (Figure 8). For the displayed checkpoints, the ordering also persists across all 32 temporal bins, each averaging 128 commits over 64 trajectories (Figure 2b–d).

Boundaries on the scored window change transfer. Zeroing the predecessor context is worse than the streaming evaluation itself, and continuing memory training for a full budget returns to where it started (Figure 6). For the training-data controls in Figure 3, we change one property at a time and hold the model, the starting weights, the step count and every hyperparameter fixed. Shortening the input sequence leaves the decoder ahead of correlated matching at all three distances, and so does placing an independent-sample seam inside the receptive field. Both have lower mean error than the stream-fine-tuned decoder at distances 7 and 9. Opening the commit window at a prepared state puts the decoder behind matching at all three, and so does closing it with a final readout (Figure 3). These interventions locate the ranking change in the treatment of the scored window: both preparation and terminal readout produce a penalty that grows with distance.

Performance depends on the buffer used in training. The buffer specifies how much of the stream a decision reads. Response time specifies how long the machine takes to produce it (Section G.1). With no buffer the cutoff is the window end. One window of lookahead follows the buffering principle used in streaming decoders (Skoric et al., 2023), where matching's error is already saturated at two windows (Figure 8). Providing one window of lookahead to the zero-buffer decoder without training it for this input moves it from below matching to above the constant predictor, at all three distances (Figure 6). A bounded-run training control inspired by the self-coordinated procedure also gives higher error than buffered matching at all three distances (Appendix C). Table 12 gives the corresponding composition-equivalent per-round quantities.

The advantage holds across tested block lengths. The fine-tuning stage gives the lowest composed-increment error at every tested block length and distance (Figure 10). Both the reversal and the stage that closes it hold at $p = 0.007$ and at $p = 0.003$, at distances 5 and 7 (Appendix D).

4.2 Transfer across decoder architectures

Fine-tuning reduces error for the released graph and transformer decoders. The transformer of Yan et al. (2026b) predicts the observable flip over the whole supplied experiment. We difference its decisions on $4T$ and $3T$ rounds with a shared start, which converts that convention into the window's net flip. On the distance-7 stream, the released prefix-differenced output remains above matching. After local-window supervision, inference uses the directly supervised head. The graph decoder of Lange et al. (2025) reads a run's triggered detectors as a graph. Its published weights exist at

Table 2: **Transfer to pre-trained graph and transformer decoders.** Per-commit errors (%) of the graph decoder of Lange et al. (2025) and the transformer of Yan et al. (2026b) on identical never-reset records within each column, at the indicated noise levels. Fine-tuned graph entries average three training runs. The released transformer uses cumulative-prefix differencing. After local-window supervision it uses the direct output (Table 18). The last row gives relative error reduction versus matching, computed from the displayed means. **Bold** marks the lowest error within each column.

	graph decoder, $p = 0.005$			transformer
Decoder	$d = 5$	$d = 7$	$d = 9$	$d = 7, p = 0.007$
pre-trained weights	20.62	30.27	37.77	40.94
causal correlated matching	10.97	14.73	18.15	21.28
after the fine-tuning stage	**9.09**	**12.96**	**16.73**	**17.34**
constant predictor	23.10	34.62	42.99	40.38
Reduction vs. matching (%)	17.1	12.0	7.8	18.5

Table 3: **Conditional parity error across distances.** Joint training versus matching at $B = 1$ on the primary set. Learned entries are means $\pm$ sample SD over three runs; each condition uses 40,000 shared shots at $p = 0.007$. All errors score J relative to q_0.

d	Joint training (%)	Joint matching (%)	Reduction (%)
5	15.73 ± 0.20	18.56	15.2
7	17.95 ± 0.12	20.94	14.3
9	21.56 ± 0.43	23.72	9.1

every distance here and decode the reset-bounded memory experiment below matching. We add one stage on interior window flips at each model's own training noise level, with the convolutional stack's supervision, target and schedule. After fine-tuning, both models outperform causal correlated matching in their tested settings: the graph decoder at all three distances and the transformer at $d = 7$ (Table 2). The transformer is fine-tuned at $p = 0.005$ and evaluated here at $p = 0.007$, jointly testing transfer of noise level and output supervision. Table 18 gives the additional output readings.

4.3 Training for conditional frame–readout composition

Lattice surgery implements joint logical Pauli measurements by changing the stabilizers measured across patch boundaries (Horsman et al., 2012). Such measurements enter logical-gate constructions and magic-state consumption (Litinski, 2019), providing an operational setting for frame–readout composition.

Two streamed patches merge for $T = d + 1$ rounds to measure $Z_A Z_B$. Let q_0 be its noiseless reference value on the noisy state before the last B windows, and q_m its value immediately before merging. On the fixed representatives, $f_A \oplus f_B = q_m \oplus q_0$. The raw parity m_{raw} combines selected pre-merge stabilizers with new merge measurements. Defining $r = m_{\mathrm{raw}} \oplus q_m$ gives the conditional correction target

$$J = m_{\mathrm{raw}} \oplus q_0 = r \oplus f_A \oplus f_B. \tag{1}$$

A prediction $\hat{J}$ corrects the raw parity to q_0 exactly when $\hat{J} = J$. Joint correlated matching and the learned systems score this same target through the merge deadline. The reference defines the scored block. No reference tap is a decoder input. Appendix F gives the measurement map and derivation. For an ideal input reference $q_\star$, the accumulated correction is $C_0 = q_0 \oplus q_\star$ and the complete correction is $E = J \oplus C_0$. The experiment scores the block contribution J.

Component rankings can reverse after composition. With the same matching-decoded readout, stream-fine-tuned frames yield a higher conditional error than matching's frame pipeline, despite their lower per-window error (Figure 14a). The relation follows from the component error bits e_A, e_B, e_r: the output fails exactly when $e_A \oplus e_B \oplus e_r = 1$. The odd-parity patterns are 100, 010, 001 and 111, so output error depends on their joint frequencies. Each marginal component error sums a different set of patterns. This explains why a component's ranking alone cannot determine the ranking of the composed prediction.

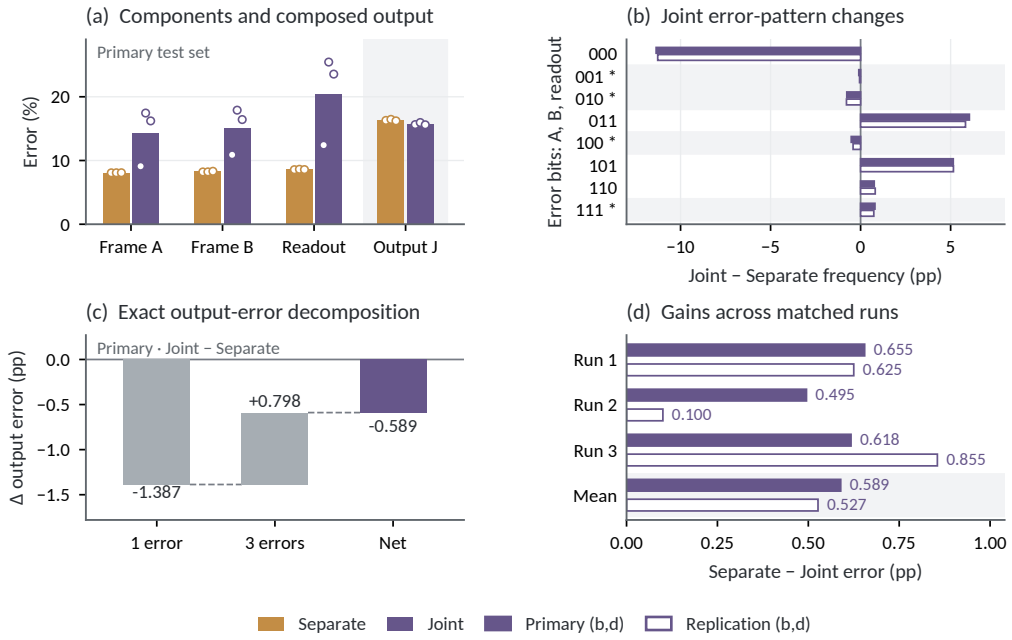


Figure 4: **Component errors and their parity composition.** (a) Component and conditional-J errors on the primary set; bars show three-run means and dots individual runs. (b) Mean joint-minus-separate changes in error-pattern frequencies; bits denote frame A, frame B and readout. Starred, shaded rows mark odd parity. (c) Mean changes in one-error and three-error events and their sum on the primary set. (d) Separate-minus-joint conditional error per matched run and its mean. Filled/open bars in (b,d) denote primary/replication sets, each with 40,000 shared shots. The same three model pairs are evaluated at $d = 5$, $B = 1$, $p = 0.007$.

Joint supervision improves accuracy on the composed target. We add a readout network on the merged lattice to the stream-fine-tuned frame decoder and combine their predictions through a differentiable parity head (Baireuther et al., 2018). Both networks are updated using binary cross-entropy (BCE). For component logits ℓ_i and $n = 2B + 1$, the head and objectives are

$$q = \frac{1 - \prod_{i=1}^{n}(1 - 2\sigma(\ell_i))}{2}, \qquad \mathcal{L}_J = \mathrm{BCE}(q, J_1), \qquad \mathcal{L}_S = \frac{1}{n}\sum_{i=1}^{n} \mathrm{BCEWithLogits}(\ell_i, a_i^{(1)}). \tag{2}$$

Here $a_i^{(1)}$ contains the $2B$ frame increments and first-round readout correction. Their parity is J_1. A measured detector parity K converts both the label and the thresholded prediction to the scored merge round. Joint training updates both networks through q. Separate training updates them through their component labels. The input cutoff and scoring rule are shared (Appendix F.1).

At $d = 5$, $B = 1$, we compare joint and separate supervision using the same networks, matched initializations and training data, and a shared optimizer configuration and step budget (Appendix F.1). Joint supervision lowers mean conditional error by 0.59 percentage points on the primary test set and 0.53 on the replication set, improving in each of the three matched runs on both sets (Appendix F.1, Figure 4d). Joint supervision increases each component's error while reducing conditional parity error across matched runs (Figure 4a,d). The joint error patterns account for this reversal (Figure 4b,c).

The trained system improves mean accuracy across distances. Table 3 compares the jointly trained system with joint correlated matching. Its mean error is lower at all three distances. The distance-9 gain is smaller. The ordering of the means is reproduced on a second test set (Appendix F.1). Longer-block evaluations extend the comparison to $B = 2, 3$ (Figure 15). These system comparisons complement the matched objective comparison at distance 5.

Table 4: **Composed predictions on device records.** Errors (%) on 12,500 shared test shots per condition. Before uses one model. After is the mean ± sample SD of three training runs. Matching uses the predictions distributed with the dataset.

Distance (grid)	Rounds	Before	After	Matching
$d = 3$ (6,7)	30	48.50	25.15 ± 0.09	25.55
$d = 5$ (4,7)	30	49.54	10.67 ± 0.14	10.53
$d = 5$ (4,7)	50	41.19	15.06 ± 0.07	15.26

4.4 Composed predictions on device data

Fine-tuning reduces composed error on device syndromes. We evaluate composed predictions on public device records (Google Quantum AI, 2023; 2024). Each run holds one logical state, and interior commits use its preceding syndrome rounds. The decoder commits at each window's end with zero buffer. The final readout supervises the parity of these decisions. Fine-tuning brings composed error from near chance to matching-level accuracy (Table 4). Zeroing the $R - 1$ predecessor rounds raises composed error; zeroing all earlier rounds gives the same result (Appendix E).

With each whole run scored as one commit, our decoders have lower error than dataset-provided correlated matching on identical shots at all three grids (Table 13, upper block).

5 Related work

Learned decoders for the surface code span convolutional (Yan et al., 2026a), recurrent (Varbanov et al., 2025), transformer (Bausch et al., 2024; Yan et al., 2026b) and graph architectures (Lange et al., 2025), and now run at the syndrome rate (Senior et al., 2025; Zhang et al., 2026b; Chamberland et al., 2026; Yang et al., 2026; Chongder, 2026; Gu et al., 2026). An early recurrent decoder separates syndrome-history processing from a terminal-readout network with a short syndrome window (Baireuther et al., 2018). Its two outputs are combined by a probabilistic XOR and trained simultaneously against final memory parity. Our composition study compares joint and separate supervision for the conditional lattice-surgery target, measuring both component and composed error.

The nearest system is the self-coordinated decoder of Zhang et al. (2026b), which commits every d rounds with a buffer of d rounds inside memory experiments and reports logical error per experiment. We commit causally and score every commit on never-reset streams, and Figure 6 tests a bounded-window training control inspired by it on our architecture and window schedule.

Finite-history recovery has a classical foundation in overlapping syndrome windows (Dennis et al., 2002). Sparse Blossom accelerates MWPM by discovering needed paths directly on the detector graph (Higgott & Gidney, 2025). Fusion Blossom divides the decoding problem and fuses subproblem solutions, with schedules for batch and stream decoding (Wu & Zhong, 2023). Sliding and parallel window matchers commit a region and buffer the rest (Skoric et al., 2023). Real-time matching decodes a live processor at the cadence we target (Google Quantum AI and Collaborators, 2025), and belief propagation meets the same budget (Müller et al., 2025). These approaches provide the computational context for our comparison of learned predictions under a common syndrome cutoff. Outside quantum computing, related work on language-model evaluation also studies how a common task-aligned fine-tuning stage affects benchmark rankings (Zhang et al., 2026a).

6 Conclusion

A decoder can beat matching on memory benchmarks yet fall behind on continuous streams. In this work, we observe this reversal in convolutional decoders at $d = 5, 7, 9$ and trace it to preparation and readout boundaries in training. Fine-tuning on interior windows restores the advantage across convolutional, graph, and transformer decoders. With one window of lookahead, the convolutional decoder reduces per-commit error by 40-45% relative to matching in simulation, under the same syndrome cutoff. In lattice surgery, better frame accuracy need not yield better conditional parity; joint supervision improves that parity over separate training at $d = 5$. Together, these findings point to a broader principle: align decoder training and evaluation with the information available when decisions are made and the outcomes that matter to the computation.

AI USE STATEMENT

The authors used Claude and ChatGPT to assist with language polishing, identifying closely related literature, improving the organization and narrative flow of text written by the authors, and limited software engineering tasks, including debugging and optimizing GPU training efficiency. All AI-assisted work was reviewed by the authors, who take full responsibility for the content, accuracy, and integrity of the manuscript.

ETHICS STATEMENT

This work studies decoders for simulated and publicly available quantum error correction data. It involves no human subjects, no personally identifying information, and no new data collection. The device syndromes are those made publicly available by Google Quantum AI and Collaborators (2025) and Google Quantum AI (2023), used under their stated terms.

REFERENCES

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, January 2018. ISSN 2521-327X. doi: 10.22331/q-2018-01-29-48. URL https://doi.org/10.22331/q-2018-01-29-48.

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

Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, et al. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, 2024.

Christopher Chamberland, Jan Olle, Muyuan Li, Scott Thornton, and Igor Baratta. Fast and accurate ai-based pre-decoders for surface codes. *arXiv preprint arXiv:2604.12841*, 2026.

Sumit Chongder. Latency-constrained hardware-aware quantum error correction co-design with adaptive confidence-gated neural decoding for the rotated surface code. *arXiv preprint arXiv:2607.05814*, 2026.

Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 09 2002. ISSN 0022-2488. doi: 10.1063/1.1499754. URL https://doi.org/10.1063/1.1499754.

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012. doi: 10.1103/PhysRevA.86.032324. URL https://link.aps.org/doi/10.1103/PhysRevA.86.032324.

Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, July 2021. ISSN 2521-327X. doi: 10.22331/q-2021-07-06-497. URL https://doi.org/10.22331/q-2021-07-06-497.

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

Google Quantum AI. Data for "quantum error correction below the surface code threshold", August 2024. URL https://doi.org/10.5281/zenodo.13273331.

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

Andi Gu, Juan Pablo Bonilla Ataides, Mikhail Lukin, and Susanne Yelin. Scalable neural decoders for practical fault-tolerant quantum computation. In *ICML 2026 AI for Science Workshop*, 2026. URL https://openreview.net/forum?id=nxnPsNyl8B.

Oscar Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching. *ACM Transactions on Quantum Computing*, 3(3):1–16, 2022.

Oscar Higgott and Craig Gidney. Sparse Blossom: correcting a million errors per core second with minimum-weight matching. *Quantum*, 9:1600, January 2025. ISSN 2521-327X. doi: 10.22331/q-2025-01-20-1600. URL https://doi.org/10.22331/q-2025-01-20-1600.

Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, dec 2012. doi: 10.1088/1367-2630/14/12/123011. URL https://doi.org/10.1088/1367-2630/14/12/123011.

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. *Phys. Rev. Res.*, 7:023181, May 2025. doi: 10.1103/PhysRevResearch.7.023181. URL https://link.aps.org/doi/10.1103/PhysRevResearch.7.023181.

Daniel Litinski. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum*, 3:128, March 2019. ISSN 2521-327X. doi: 10.22331/q-2019-03-05-128. URL https://doi.org/10.22331/q-2019-03-05-128.

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.

Andrew W Senior, Thomas Edlich, Francisco JH Heras, Lei M Zhang, Oscar Higgott, James S Spencer, Taylor Applebaum, Sam Blackwell, Justin Ledford, Akvilė Žemgulytė, et al. A scalable and real-time neural decoder for topological quantum codes. *arXiv preprint arXiv:2512.07737*, 2025.

Luka Skoric, Dan E Browne, Kenton M Barnes, Neil I Gillespie, and Earl T Campbell. Parallel window decoding enables scalable fault tolerant quantum computation. *Nature Communications*, 14(1):7040, 2023.

Boris M. Varbanov, Marc Serra-Peralta, David Byfield, and Barbara M. Terhal. Neural network decoder for near-term surface-code experiments. *Phys. Rev. Res.*, 7:013029, Jan 2025. doi: 10.1103/PhysRevResearch.7.013029. URL https://link.aps.org/doi/10.1103/PhysRevResearch.7.013029.

Yue Wu and Lin Zhong. Fusion blossom: Fast mwpm decoders for qec. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 01, pp. 928–938, 2023. doi: 10.1109/QCE57702.2023.00107.

Ge Yan, SHANCHUAN LI, and Yuxuan Du. Rethink the role of neural decoders in quantum error correction. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=EnZFaqg7hx.

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

Xiaohan Yang, Xuandong Sun, Zhiyi Wu, Jiawei Zhang, Ji Jiang, Xiayu Linpeng, Yuxuan Zhou, Ji Chu, Jingjing Niu, Youpeng Zhong, et al. Real-time surface-code error correction using an fpga-based neural-network decoder. *arXiv preprint arXiv:2605.04892*, 2026.

Guanhua Zhang, Ricardo Dominguez-Olmedo, and Moritz Hardt. Train-before-test harmonizes language model rankings. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 74776–74808, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/79a97ba70ee6aff126f3090b10ad6702-Paper-Conference.pdf.

Kai Zhang, Zhengzhong Yi, Shaojun Guo, Linghang Kong, Situ Wang, Xiaoyu Zhan, Tan He, Weiping Lin, Tao Jiang, Dongxin Gao, et al. Learning to decode in parallel: Self-coordinating neural network for real-time quantum error correction. *arXiv preprint arXiv:2601.09921*, 2026b.

A TEMPORAL RECEPTIVE FIELD AND STREAMING EVALUATION

Section 3.2 derives the required history length from the computation graph. This appendix defines the objects, proves the span and streaming equivalence, and verifies these properties numerically.

A.1 SETTING AND THE RECEPTIVE FIELD OF A CAUSAL STACK

A stream is $x \in \{0,1\}^{\mathbb{Z}\times V}$ over a finite nonempty site set V, with frame x_t. A field is $u : \mathbb{Z} \to \mathbb{R}^{V\times C}$, with finite channel count C. Window length T and dilations d_ℓ are positive integers. Operators map between the appropriate field spaces, with fixed parameters and evaluation-mode normalisation statistics. P denotes a temporal pointwise operator, $(Pu)_t = f_t(u_t)$, where each f_t is fixed independently of x; the usual case is one fixed f for every t. Any known lattice-occupancy schedule is treated as a fixed exogenous input and aligned at the same absolute rounds across evaluations. Embeddings, $1\times1\times1$ convolutions, fixed-statistic normalisation and element-wise nonlinearities are of this form. The causal temporal convolution with dilation d is

$$(K_d u)_t = W_0 * u_t + W_1 * u_{t-d} + W_2 * u_{t-2d}, \tag{3}$$

each W_j acting on sites and channels only.

Definition 1 (receptive field)

$$\rho(A) = \min\{r \in \mathbb{N}_0 : (Au)_t \textit{ is determined by } u_{t-r}, \dots, u_t \textit{ for all } u, t\}.$$

So $\rho(P) = 0$ and $\rho(K_d) \le 2d$.

Lemma 1 *$\rho(A \circ B) \le \rho(A) + \rho(B)$, and $\rho(I + A) \le \rho(A)$.*

Proof. $(ABu)_t$ is determined by $(Bu)_s$ for $s \in [t - \rho(A), t]$, and each such slice by $u_{s'}$ for $s' \in [s - \rho(B), s]$. The union is $[t - \rho(A) - \rho(B), t]$. The sum case follows because both terms are determined by the larger interval. □

Definition 2 (network) *With pointwise maps E, P_0 and $P_{\ell,\mathrm{in}}, P_{\ell,\mathrm{out}}$ per block,*

$$F = S \circ P_0 \circ B_L \circ \cdots \circ B_1 \circ E, \qquad B_\ell = I + P_{\ell,\mathrm{out}} \circ K_{d_\ell} \circ P_{\ell,\mathrm{in}}, \tag{4}$$

where the scatter $S = K_1$. Distinct occurrences of K_d carry separate sets of taps.

Definition 3 (commit) *Fix c and masks $M_t \subseteq V$ for $t \in [c - T + 1, c]$, independent of x, with $|M| := \sum_{t=c-T+1}^{c} |M_t| > 0$ and every M_t nonempty in the deployed configuration. The commit score is*

$$\hat{y}(c) = h\Big(\tfrac{1}{|M|} \textstyle\sum_{t=c-T+1}^{c} \sum_{v\in M_t} (Fx)_{t,v}\Big). \tag{5}$$

Here $h : \mathbb{R}^C \to \mathbb{R}$ returns the continuous output, a logit in the deployed network; its binary prediction is $\hat{b}(c) = \mathbf{1}[\hat{y}(c) > 0]$. Throughout this appendix, support and exact score reproduction refer to $\hat{y}(c)$.

Write $D := \sum_{\ell=1}^{L} d_\ell$, $R := 2D + 3$ and $\ell_{\max} := R + T - 2$ for the candidate maximum commit lag.

Hypothesis N. At commit c there exist two streams differing only in one coordinate of the frame at time $c - \ell_{\max}$ whose commit scores differ. Hypothesis N depends on the trained parameters, the masks and the head, not on the architecture alone. A nonzero change in the score after flipping one input coordinate at that lag supplies such a pair.

Theorem 1 (support bound and conditional attainment) *The network satisfies $\rho(F) \le R - 1$, and $\hat{y}(c)$ is determined by $x_{c-\ell_{\max}}, \dots, x_c$. Under Hypothesis N the commit's maximum lag $\ell_{\max}$ is attained and $\rho(F) = R - 1$.*

Proof. Lemma 1 gives $\rho(B_\ell) \le 2d_\ell$ and $\rho(S) \le 2$, so $\rho(F) \le 2 + \sum_\ell 2d_\ell = R - 1$. The first round of the pooling window is $t_* = c - T + 1$, whose determination extends to $t_* - (R-1)$, which is lag $(T-1) + (R-1) = R + T - 2$. For attainment let x, x' be Hypothesis N's witnesses, differing only at round $t_* - (R-1)$. That round lies at lag $R-1$ before t_* and further before every later pooled round, so by the bound it can change $(Fx)_t$ only at $t = t_*$. Since $\hat{y}(c)$ differs, that field slice must differ, so $\rho(F) \ge R - 1$. □

Attainment identifies the largest influential lag, not dependence on every intermediate lag or site. The upper bound also holds for $\hat{b}(c)$; attainment for the binary prediction requires a witness that changes $\hat{b}(c)$ itself.

With unit dilations $R = 2L + 3$. For the deployed family $L = d$ and $T = d + 1$, so $R - 1 = 2d + 2 = 2T$. That is the prescription of Section 3.2.

A.2 Resetting and carrying the network state

Definition 4 (causal cut) *Let $u_{\ell,t}$ be the input to K_{d_ℓ} after $P_{\ell,\mathrm{in}}$, and let v_t be the scatter input after P_0. The cut at c is the ordered tuple*

$$\Sigma(c) = \left(\left((u_{\ell,t})_{t=c-2d_\ell+1}^{c} \right)_{\ell=1}^{L}, (v_{c-1}, v_c) \right) \in \mathcal{S} := \left(\prod_{\ell=1}^{L} \mathcal{U}_\ell^{2d_\ell} \right) \times \mathcal{U}_S^2,$$

where $\mathcal{U}_\ell$ and $\mathcal{U}_S$ are the respective convolution input spaces. Thus each block stores its $2d_\ell$ most recent convolution inputs and the scatter stores two, giving $\sum_\ell 2d_\ell + 2 = R - 1$ field slices in total. The stored tensors have the channel and lattice dimensions of their respective input spaces, which determine the number of stored scalars.

Both evaluations supply the T window frames $[c - T + 1, c]$. Write $G_c(s; x_{c-T+1:c})$ for the window score with initial state $s \in \mathcal{S}$ and the fixed masks above. Zero-state evaluation is $G_c(0_{\mathcal{S}}; x_{c-T+1:c})$; carried-state evaluation is $G_c(\Sigma(c-T); x_{c-T+1:c})$ and equals the bi-infinite score. Zero feature queues generally differ from the state produced by all-zero detector frames, since learned embeddings and pointwise maps need not map zero to zero.

Proposition 1 *The two evaluations at the same commit differ only in the initial state assignment, with the cut $\Sigma(c-T)$ at the window's opening boundary replaced by $0_{\mathcal{S}}$. Under Hypothesis N they are distinct functions of the stream.*

Proof. Both run the same F on the same T frames and differ only in the initial queues, and those queues plus the window frames determine every subsequent slice. For distinctness take Hypothesis N's witnesses, which differ only at lag $\ell_{\max}$. Since $\ell_{\max} - T = R - 2 \ge 0$ they agree on the window frames, so the reset scores agree while the carried scores differ. At least one witness therefore has different scores under the two evaluations. □

A.3 Streaming and replay evaluation

The replay evaluator at c runs the finite evaluation on the window plus the $R-1$ preceding rounds from zero queues. The streaming implementation processes contiguous windows in order, carrying the queues of Definition 4 across each boundary. It starts from zero queues at round b_0, with commits $c_k = b_0 + (k+1)T - 1$ for $k \ge 0$.

Theorem 2 (equivalence of streaming and replay evaluation) *At any commit whose window is preceded by at least $R-1$ processed rounds, $c - T + 1 - b_0 \ge R - 1$, the streaming implementation, the replay evaluator and the bi-infinite network return the same commit score and hence the same binary prediction.*

Proof. Set $a = c - T + 1 - (R-1)$. The precondition gives $b_0 \le a$. All three computations receive identical frames from a onward. Their queue states immediately before round a may differ. We show inductively that the output of $B_\ell \circ \cdots \circ B_1 \circ E$ agrees from $\tau_\ell := a + 2\sum_{j=1}^{\ell} d_j$ onward.

For the first block, $u_{1,t} = (P_{1,\mathrm{in}} E x)_t$ depends only on the current frame, so these inputs agree for $t \ge a$. All three inputs at $t, t - d_1, t - 2d_1$ therefore agree for $t \ge a + 2d_1$, as does the residual

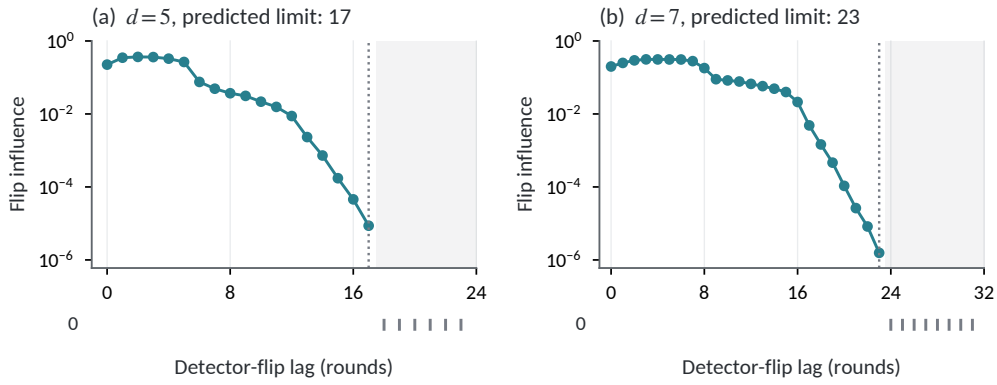


Figure 5: **Measured input influence and architectural reach.** (a,b) Mean absolute change in the sigmoid of the commit score after one detector flip, versus lag, for one trained decoder per distance at $d = 5, 7$, using 2,000 probe shots per lag at $p = 0.007$. Positive values use a logarithmic scale. Gray strips show exact zeros. Dotted lines mark the predicted last reachable lags, 17 and 23 rounds, respectively.

branch. The first block's outputs agree from τ_1. If the preceding stack agrees from $\tau_{\ell-1}$, the inputs $u_{\ell,t}$ agree from that time, and all three delayed inputs to K_{d_ℓ} agree for $t \geq \tau_{\ell-1} + 2d_\ell = \tau_\ell$. The residual input and pointwise output map then give the induction step. Finally, P_0 adds no delay and the scatter adds two rounds, so F agrees from $a + 2D + 2 = a + R - 1 = c - T + 1$. All three pool identical slices over the whole window and apply the same h. □

The equivalence holds for all fixed parameters, including those that do not satisfy Hypothesis N. Appendix G reports the comparisons between implementations at their specified numerical precision.

Corollary 1 (minimal replay) *Fix a commit satisfying Theorem 2's precondition. Among deterministic evaluators that see the window plus the q immediately preceding frames, with no carried state or other side information, and under Hypothesis N, no $q < R - 1$ reproduces the streaming score on all streams, while $q = R - 1$ suffices.*

Proof. Hypothesis N supplies streams x, x' differing only at $c - \ell_{\max} = c - T + 1 - (R - 1)$. For $q < R - 1$ this frame is strictly earlier than the first visible frame $c - T + 1 - q$. The evaluator receives identical inputs on x, x' and must return the same value, whereas their streaming scores differ by Theorem 2. It cannot reproduce both. Theorem 2 gives sufficiency for $q = R - 1$. □

Without nondegeneracy, this lower bound need not hold; for example, a constant head can be reproduced without any predecessor frames. The same lower-bound argument applies to binary predictions when the witness changes $\hat{b}(c)$.

A.4 Numerical verification

For dilation schedules $[1]$, $[1, 1]$, $[1, 2]$, $[1, 2, 4]$ and $[2, 3, 1]$ the symbolic delay polynomial has degree 4, 6, 8, 16 and 14, the network's impulse response has its last nonzero lag at exactly those values and is zero beyond. For the four schedules $[1]$, $[1, 2]$, $[1, 2, 1]$ and $[2, 3, 1]$, flipping one input at the last reachable lag changes the commit score by 9.4×10^{-2}, 6.7×10^{-3}, 2.5×10^{-2} and 2.1×10^{-2}, and one lag past it by exactly zero in all four. The streaming implementation against the replay evaluator is bitwise identical on 38 of 38 commits, and dropping the scatter's two slices from the carried cut breaks the equality on 31 of them, with differences up to 0.70. In the trained decoders the influence probe of Figure 5 shows influence of 8.7×10^{-6} at lag 17 for $d = 5$ and 1.6×10^{-6} at lag 23 for $d = 7$, the respective last reachable lags. Beyond these lags, flipped and unflipped outputs agree bit for bit on all 2,000 shots at each distance. Since sigmoid is strictly monotone, a nonzero probability difference also establishes score dependence. These probes test attainment for the evaluated weights.

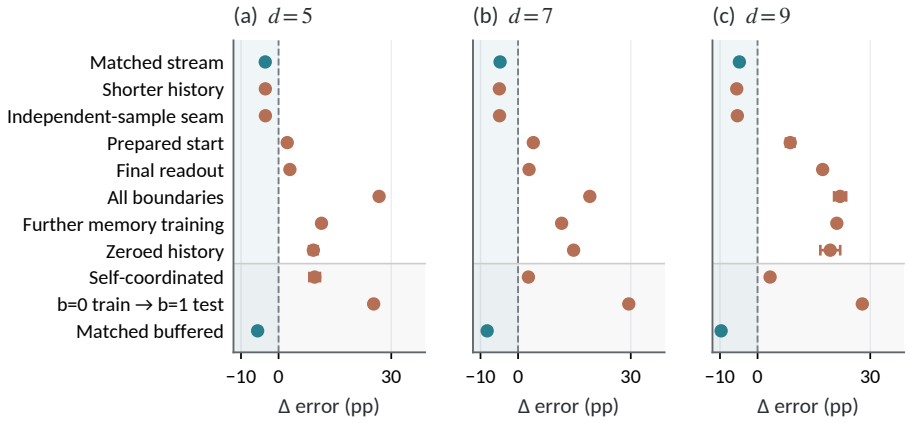


Figure 6: **Complete training and evaluation controls.** (a–c) Per-commit error minus corresponding matching at $d = 5, 7, 9$, in percentage points, for eleven conditions at $p = 0.007$. Points and error bars show three-run means and sample SD. The first eight rows use no buffer. The shaded final three use one window. Zero marks matching with the corresponding buffer.

B Effect of training history length

Section 3.2 states that $R - 1$ bounds the predecessor history a causal stack can use. This appendix measures the effect of using less history.

B.1 The single-factor controls

Every training-data control in Figure 6 runs the added stage item for item at its own distance, from the same stitched initialization within that control family, at the same width, for the same 40,000 steps, batch and learning rate, and changes one property of the training stream. All are scored on the stream of Section 4.1, on the same 262,144 commits, against the same causal matcher.

The sequence-length entry feeds three windows, whereas the fine-tuning stage feeds four. Thus, the input history starts exactly at the far edge of the commit's receptive field because $R - 1 = 2T$ at all three distances. That entry and the seam entry both retain lower mean error than causal correlated matching at all three distances, whether the boundary is at the edge of the receptive field or inside it. The prepared state acts when it opens the committed window itself.

B.2 Exact training and inference protocols

Use zero-based grid slices and half-open intervals. Let z_k be the reference bit after kT rounds, with $z_0 = 0$ for the prepared initial state, and u the final transversal-readout bit. A training example uses a continuous circuit with one guard window beyond the input, except when terminal readout is the intervention. The guard and reference taps are never network inputs. For a pooled window w, the head averages the causal features on slices $[wT, (w + 1)T)$ over occupied lattice sites.

Training condition	Fed slices	Pooled slices	Label
Interior stream	$[0, 4T)$	$[3T, 4T)$	$z_4 \oplus z_3$
Shorter sequence	$[0, 3T)$	$[2T, 3T)$	$z_3 \oplus z_2$
Prepared start	$[0, 4T)$	$[0, T)$	z_1
Independent seam	$[0, 4T)$	$[3T, 4T)$	$z_4 \oplus z_3$
Final readout	$[1, 4T + 1)$	$[3T, 4T)$ local	$u \oplus z_3$
One-window buffer	$[0, 5T)$	$[4T, 5T)$	$z_4 \oplus z_3$

In the seam condition, slices $[0, 2T)$ are replaced by the corresponding slices of an independent sample. Slices $[2T, 4T)$ and the label come from the scored sample. Detector values are concatenated without recomputing a cross-seam difference. This independent-sample seam is located at $2T$,

inside the $[T, 3T)$ predecessor span of the final window. In the terminal condition, the circuit ends at $4T$. Dropping grid slice 0 and including terminal slice $4T$ keeps $4T$ input slices. Its last pooled local window therefore includes the terminal detector layer. The network occupancy mask is the first $4T$ slices of the circuit lattice mask, as in the training implementation. Prepared-start training supervises the first window of the same input circuit. Causality excludes the later input slices from this prediction. Each call starts with zero causal padding and no carried activations from another training example.

The boundary-control family and buffer-trained models share the existing stitched initialization within each distance, as listed in the checkpoint map. The headline zero-buffer fine-tuning rows instead use the corresponding 240,000-step memory checkpoints. The control family holds 40,000 AdamW steps, batch 256, learning rate 2×10^{-4}, weight decay 10^{-2} and exponential moving average (EMA) decay of 0.9998 fixed, across three independent training runs. Inserting a preparation or readout changes the specified detector/label pair. These controls measure the effect of each boundary treatment, including its detector statistics and label distribution.

For evaluation of target window j, a model with C input windows and lookahead b receives slices $[(j - C + b)T, (j + b)T)$ and pools its final window, while the label remains $z_j \oplus z_{j-1}$. Memory-trained controls use $C = 3$. Zero-buffer fine-tuned models use $C = 4$. Buffer-trained models use $C = 5$, $b = 1$. The buffered Memory row reuses memory weights with shifted pooling, without retraining on the earlier target. It is an input/readout-mismatch control. The buffered fine-tuned-versus-matching comparison uses the trained $b = 1$ target and matching with the same cutoff. The unbuffered ranking reversal is established separately with $b = 0$.

B.3 Matched history at distance 5

We train checkpoints with 0, 1, 2 and 3 windows of real predecessor history and score them on the never-reset stream of Section 4.1. Figure 7 shows the result. Zero windows performs at the constant-predictor level. Every nonzero history condition beats correlated matching, and one window has the lowest mean error among the tested history lengths. These are a separate block of weights from the stream-fine-tuned entry of Table 5, which supplies $3T$. The prepared-start control of Figure 3a commits the first window of its circuit, and its error lies between the zero- and one-window conditions of Figure 3b. A prepared window supplies its opening state directly, whereas a stream supplies it through history.

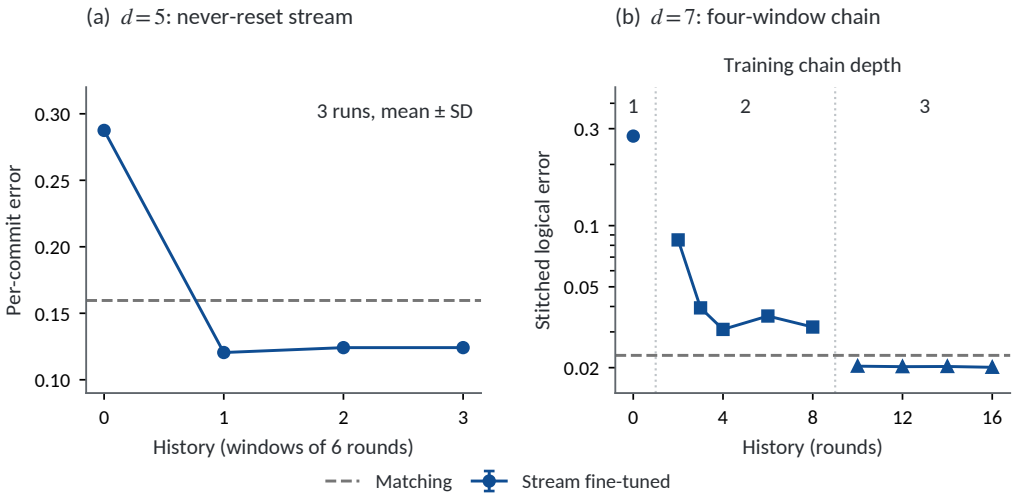


Figure 7: **History length and training-chain depth.** (a) Per-commit error versus predecessor windows on the never-reset $d = 5$ stream. Points show three-run means $\pm$ sample SD. (b) Stitched logical error at $d = 7$ versus predecessor rounds, with 200,000 shots per point. Colors identify training-chain depths: 1 at $h = 0$, 2 at $2 \le h \le 8$, and 3 at $h \ge 10$. Dashed lines mark matching.

Table 5: **Per-commit error with and without a buffer.** Each decoder scores 262,144 commits of $T = d + 1$ rounds per distance at $p = 0.007$. The upper block uses one buffer window. The lower block uses none. Learned entries are means ± SD over three training runs. **Bold** marks the lowest error within each block and distance.

Decoder	$d = 5$	$d = 7$	$d = 9$
one window of buffer			
correlated matching	0.1390	0.1812	0.2153
memory-trained	0.3457 ± 0.0116	0.4491 ± 0.0086	0.4829 ± 0.0103
stream-fine-tuned	$\mathbf{0.0833 \pm 0.0002}$	$\mathbf{0.0990 \pm 0.0002}$	$\mathbf{0.1183 \pm 0.0004}$
no buffer			
causal correlated matching	0.1596	0.2128	0.2576
memory-trained	0.2380 ± 0.0226	0.3412 ± 0.0083	0.4407 ± 0.0239
stream-fine-tuned	$\mathbf{0.1242 \pm 0.0003}$	$\mathbf{0.1650 \pm 0.0035}$	$\mathbf{0.2092 \pm 0.0064}$
constant predictor	0.2875	0.4038	0.4681

B.4 Training-history fill at distance 7

Checkpoints trained with h rounds of real history are indexed by $\rho = h/R$, normalized to the per-round receptive-field length $R = 17$ at distance 7, and are scored on stitched streams against correlated matching on the same shots. Figure 7 holds the ten training points. The loss falls steeply up to $\rho = 0.24$, moves within a narrow band from there to $\rho = 0.47$, and falls again to a floor at $\rho = 0.59$ and beyond. The sign flips between those last two points, where the training chain also changes from two experiments to three, so this block orders the joint history and chain-depth configurations.

C The never-reset streams

Construction. For each of distances 5, 7 and 9 we generate 64 independent trajectories of 26,118, 34,824 and 43,530 rounds of circuit-level depolarizing noise at physical error rate 0.007, with no state preparation and no final readout anywhere inside a trajectory. Each trajectory is divided into contiguous windows of $T = d + 1$ rounds. The first 256 windows of each trajectory are discarded before scoring and the next 4,096 are scored, which gives 262,144 commits per distance. The fine-tuning stage draws its own circuits from the same generator at every optimizer step.

Labels. The observable flip a commit is scored against comes from noiseless reference taps, auxiliary readouts of the simulator that enter only the label and never the input. Removing the taps and projecting their observables through the detector error models of the two circuits agrees on every mechanism to 5×10^{-18} in probability. The taps preserve the circuit's error mechanisms.

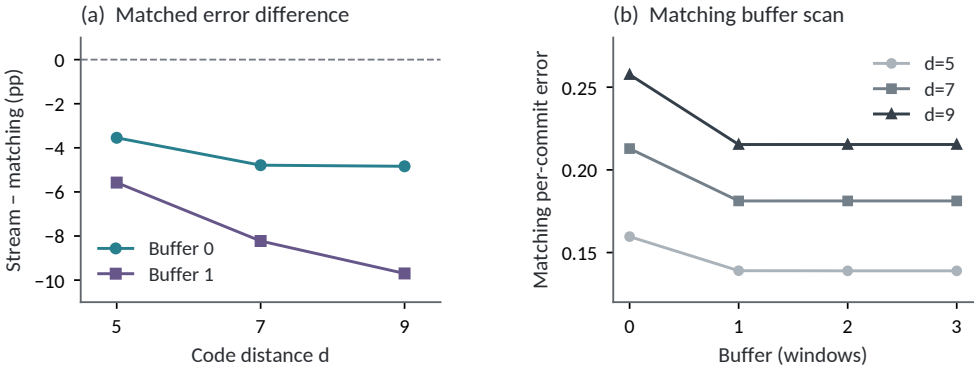
What a deadline costs. Some faults flip the logical observable and fire no detector by the deadline, $2d$ of them in a window of T rounds, and one window of buffer empties that single-fault set. The information constraint is shared by all compared decoders. Faults on opposite sides of the reference boundary can share syndrome and have different increment labels even with lookahead. All comparisons use the same fixed-representative target.

The baseline, step by step. Correlated matching commits window j from the detectors of rounds $[c - W, c)$, where $c = (j + 1)T$ is the commit and $W = (\text{band} + 1)T$ with a band of 16 windows. With one window of buffer the interval closes at $c + T$ instead. The graph is built once per distance:

1. Flatten the never-reset circuit's detector error model at that distance and noise, with decomposition enabled.
2. Keep the band's detectors and drop the rest, leaving a mechanism that touched a dropped detector with the detectors it still has, so a fault crossing the left edge becomes a boundary mechanism, which is what an open edge is.
3. Give each mechanism the commit's observable: the interior taps are cumulative frames, so window j's flip is $L_j \oplus L_{j-1}$, and a mechanism carries it when exactly one of those two taps is among its observables.

4. Drop a mechanism whose remaining detectors are empty and whose flip bit is zero, and keep it when that bit is one, the faults of Section 3.1 that flip the observable and fire no detector by the deadline.
5. Build the matching graph from the rewritten model with correlations enabled, the decomposition preserved.

The burn-in exceeds the band, so every scored commit's interval is bulk rounds and one graph serves all of them up to a shift of detector indices. Nothing is carried between commits: each is decoded from its own interval, which is what a decoder answering at the deadline has. Widening the band from $16T$ to $32T$ changes none of the 262,144 decisions at any distance. A matcher built independently from the full history back to round 0, using its own circuit and detector error model with no band or rewritten left edge, agrees with the banded one on 6,016 of 6,016 decisions over 94 commits spread across the three distances.

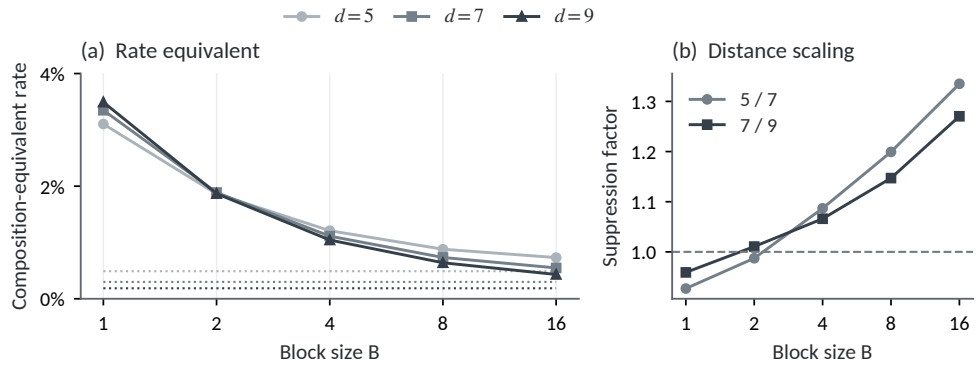


(a): three-run mean difference. Both panels: p=.007, 64 trajectories.

Figure 8: **Decoder performance versus buffer budget.** (a) Mean per-commit error difference between stream fine-tuning and matching with the same buffer, over three training runs. Differences are in percentage points. (b) Matching error with 0–3 buffer windows at $d = 5, 7, 9$. Both panels use $p = 0.007$ and the same 64 trajectories.

Figure 9 traces the commit interval for correlated matching on the same streams.

Belief propagation on the same commits. Belief propagation has also been studied at the syndrome rate (Müller et al., 2025). We run plain belief propagation over 30 iterations and a variant with



64 trajectories. Dotted lines: memory references.

Figure 9: **Matching performance versus block length.** (a) Effective per-round error at $d = 5, 7, 9$ for blocks of $B = 1, 2, 4, 8, 16$ windows. Dotted lines mark memory-experiment references. (b) Suppression factors, rate(d)/rate$(d + 2)$. The dashed line marks one. Rates are composition-equivalent quantities, evaluated at $p = 0.007$ over 64 trajectories.

localised statistics decoding. Both use the same detector error model, history band, labels, and test commits as matching.

Table 6: **Classical and learned decoding on never-reset streams.** Per-commit errors at $p = 0.007$. Belief propagation uses 30 iterations and the matcher's history band and labels, scoring 1,024 commits on each of 64 trajectories. Learned entries are the three-run means from Table 5. **Bold** marks the lowest error within each column.

per-window error, $p = 0.007$	$d = 5$	$d = 7$	$d = 9$
belief propagation	0.2081	0.3027	0.3791
belief propagation with localised statistics	0.1541	0.2125	0.2675
correlated matching	0.1596	0.2128	0.2576
trained on memory experiments	0.2380	0.3412	0.4407
after the fine-tuning stage	**0.1242**	**0.1650**	**0.2092**

Plain belief propagation has a higher error than the matcher at every distance. The localised variant is close to it, lower at distances 5 and 7 and higher at 9. The stream-fine-tuned decoder is below both everywhere (Table 6).

The self-coordinated procedure on these streams. The procedure of Zhang et al. (2026b) generates $3d$ to $5d$ rounds of a memory experiment, splits them into three to five decoding windows and mixes the window types into one dataset, each window a buffer, a core and a buffer of d rounds with the simulator's local labels, and commits the core. We implement a training-data control inspired by this procedure on our architecture and window schedule. The optimizer settings and training budget follow the added stage, and the data use a bounded-run distribution: a bounded run of four windows, the committed window drawn uniformly across it so a preparation opens some and a readout closes others, and the head pooled one window past the core. Across three training runs, the ranges of per-commit error (maximum minus minimum) are 0.028 at distance 5 and 0.003 and 0.002 at distances 7 and 9.

C.1 Published evaluation protocols

Table 7 places the learned decoders this paper cites on the same axes. Each is trained and scored on runs that are reset before they begin and read out when they end. Two rows commit inside a run, the self-coordinated decoder with a buffer of d rounds, on windows cut from bounded runs with simulator labels, and the FPGA decoder every round at distance 3. The pre-decoder's 1,000-round windows and the real-time matcher's million-round run are timed, not scored. None of the cited studies scores a decoder on a never-reset run. Table 8 separates, for the two nearest protocols and for this work, the training context from the supervised output and the evaluation target.

D Supplementary measurements

D.1 A second physical error rate

At distances 5 and 7, decoders trained on memory experiments lose to matching on never-reset streams at both noise levels. The fine-tuning stage restores their lead at both (Table 9).

The fine-tuning stage gives a larger relative error reduction against matching at the lower noise level at both distances.

D.2 The stitched stress test

One real predecessor inverts the ordering. A lower-cost test than a never-reset stream places one real experiment behind another. The committed shot arrives with a predecessor's syndrome stitched before it by detector coordinates, and its own readout scores it (Figure 11b). We call that condition carried and the same shot decoded alone isolated, and report logical error per experiment, the published quantity. Isolated, our memory-trained decoders beat correlated matching at distances 5 through 11. In the carried condition, their error is $8\times$ the isolated error at $d = 5$ and $44\times$ at $d = 11$, and they lose to matching at all four distances. Masking the predecessor does not return the

Table 7: **Decoder reset protocols and commit cadences.** A cross denotes scored runs bounded by preparation and readout. Columns give rounds per training example, longest scored run, and decisions scored per run. Superscript r denotes a repetition code.

Decoder	**Never reset**	Rounds trained	Longest scored	Commits
AlphaQubit (Bausch et al., 2024)	✗	25	10^5	1
AlphaQubit 2 (Senior et al., 2025)	✗	up to 250	10^6	1
Convolutional (Yan et al., 2026a)	✗	d	d	1
Foundation transformer (Yan et al., 2026b)	✗	d	d	1
Recurrent (Varbanov et al., 2025)	✗	up to 37	600	1
Graph network (Lange et al., 2025)	✗	d	50^{r}	1
Cascade (Gu et al., 2026)	✗	d	d	1
Latency-aware (Chongder, 2026)	✗	d	d	1
FPGA recurrent (Yang et al., 2026)	✗	up to 20	20	every round
Self-coordinated (Zhang et al., 2026b)	✗	$3d$ windows	1,400	every d rounds
Pre-decoder (Chamberland et al., 2026)	✗	9–17	31	matching decides
Real-time matching	✗	—	250	1
This work	✓	streams	43,530	4,096 per run

Table 8: **Training boundaries, supervision and evaluation.** Training context, supervised output and evaluation target for selected neural decoding protocols.

Protocol	Training boundary/context	Supervised output	Evaluation target
Recurrent memory (Baireuther et al., 2018)	Prepared sequence, history and terminal-readout networks	Final parity through combined network outputs	Logical parity of a memory experiment
Self-coordinated (Zhang et al., 2026b)	Buffered windows cut from bounded memory runs	Local labels in simulation and global parity on device runs	Logical error per memory experiment
This work: stream	Interior windows with real predecessor history	Fixed-representative window increment	Error of each never-reset commit
This work: composition	Streamed patches followed by a merge	Conditional parity J versus separate component labels	Conditional parity error through the merge deadline

Table 9: **Decoding at two physical error rates.** Upper rows report per-commit errors on 262,144 never-reset commits per distance. Lower rows report isolated-memory errors at $d = 5$. Entries with $\pm$ give means and SD over three runs. At $p = 0.003$, stream-fine-tuned entries give three-run min–max ranges, and memory-trained stream entries use one run. **Bold** marks the lowest error in each row.

	p	matching	memory-trained	after the fine-tuning stage
$d = 5$	0.007	0.1596	0.2380 ± 0.0226	$\mathbf{0.1242 \pm 0.0003}$
	0.003	0.06308	0.10427	**0.03865–0.03883**
$d = 7$	0.007	0.2128	0.3412 ± 0.0083	$\mathbf{0.1650 \pm 0.0035}$
	0.003	0.08854	0.15524	**0.05325–0.05331**
$d = 5$, isolated	0.007	0.02865	$\mathbf{0.02510 \pm 0.00020}$	—
	0.003	0.002150	$\mathbf{0.00204 \pm 0.00004}$	—

decoder to its isolated reading. A recurrent decoder (Varbanov et al., 2025), trained with its authors' package over 1 to 37 rounds and behind matching already in isolation, has 6× its isolated error at $d = 5$ when carried. Decoders trained on real device data invert the same way at distance 5 on real device shots, on both the 2023 memory dataset and the below-threshold dataset. The stitched-stage decoders win the same test at all four simulated distances. On device shots, stitched adaptation recovers near-isolated accuracy.

Boundary effects within the scored block. The join between the two experiments is a readout followed by a preparation. On device shots that join reaches $R - 1$ rounds into a committed block of 31 rounds. It covers 26% of the block at $d = 3$ and 39% at $d = 5$, and the decoder keeps its lead at the first coverage and loses it at the second.

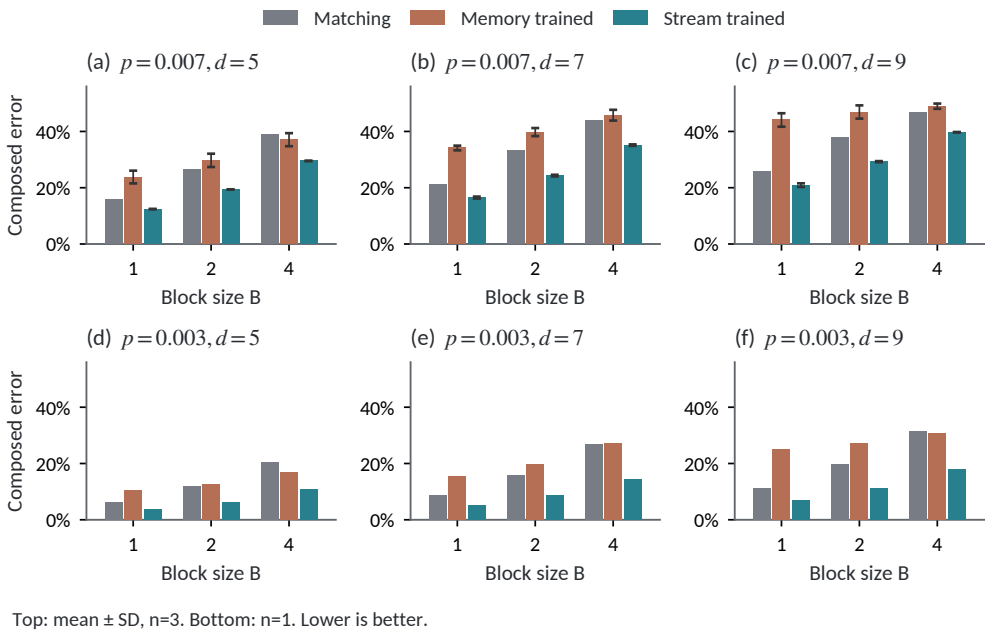


Figure 10: **Composed-frame error across block sizes and noise levels.** Matching, memory-trained and stream-fine-tuned decoders are scored on the same targets after $B = 1, 2, 4$ windows. Columns correspond to $d = 5, 7, 9$. (a–c) $p = 0.007$, with neural means $\pm$ sample SD over three training runs. (d–f) $p = 0.003$, with one training run per neural condition.

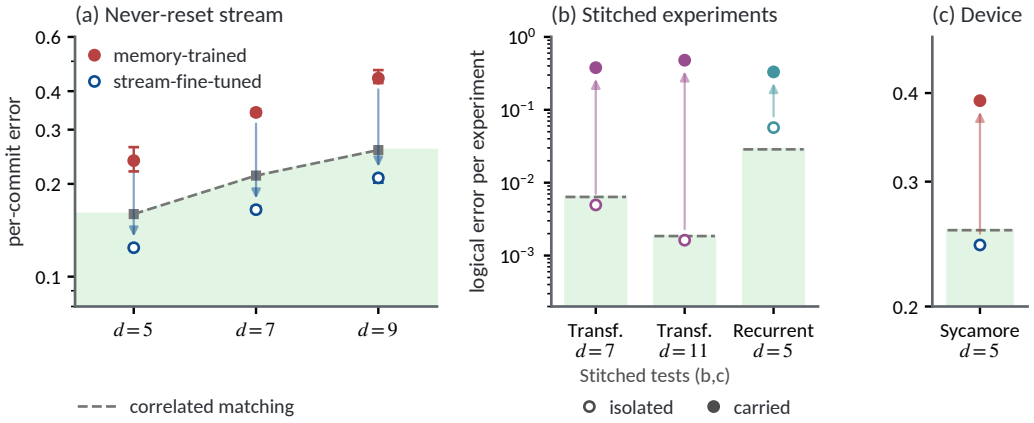


Figure 11: **Decoding never-reset streams and stitched experiments.** (a) Per-commit error before (filled) and after (open) stream fine-tuning. Points show three-run means and error bars span the observed min–max. (b,c) Logical error per experiment for pre-trained transformer and recurrent decoders (Yan et al., 2026b; Varbanov et al., 2025), and the device-trained decoder (Google Quantum AI, 2023), decoded alone (open) or with a predecessor (filled). The carried transformer output is scored against the second experiment's label. Dashed lines mark matching on identical records. Green shading marks lower error.

Table 11 gives the isolated and carried error rates for four simulated distances, pre-trained decoders and device datasets. The isolated column of Table 1 uses the final 240,000-step checkpoint evaluations. This stitched study uses a separate shot set with its own matching scores.

The transformer's carried entries score its whole-input prediction against the second experiment's readout. The supervised direct reading of the never-reset stream falls below matching after fine-tuning, including at the model's own noise level (Table 18).

After stitched fine-tuning on device shots, all three decoders reach the dataset-provided matcher's accuracy and beat it at distance 3, and carry a predecessor at close to their isolated error (Table 10). Every comparison uses identical shots.

Table 10: **Stitched evaluation on held-out device shots.** Columns give the boundary's share of the scored block, carried/isolated error ratios before and after stitched training, and the final error minus dataset matching on identical shots. BT denotes the below-threshold dataset.

	the boundary's	carried / isolated		after stage vs. matcher
dataset	share of the block	before	after	difference
2023, $d = 5$	—	1.60	1.04	$+0.002$
BT, $d = 3$	26 %	1.006	1.009	-0.008
BT, $d = 5$	39 %	1.13	1.027	$+0.001$

Table 11: **Logical error in isolated and stitched experiments.** Isolated decodes a shot alone. Carried prepends a real predecessor. Masked zeroes that predecessor's input. Simulated learned entries are means $\pm$ SD over three training runs at $d = 5, 7, 9$ and two at $d = 11$. Pre-trained entries use the authors' weights. BT denotes below-threshold device data.

	matching	isolated	carried	carried, masked
this work, $d = 5$	0.0286	0.0251 ± 0.0002	0.2129 ± 0.0388	0.1436 ± 0.0509
this work, $d = 7$	0.0235	0.0196 ± 0.0003	0.3113 ± 0.0433	0.2823 ± 0.0895
this work, $d = 9$	0.0183	0.0154 ± 0.0004	0.4312 ± 0.0577	0.4406 ± 0.1125
this work, $d = 11$	0.0142	0.0115 ± 0.0009	0.5078 ± 0.0248	0.5910 ± 0.0612
stitched stage, $d = 5$	0.0286	0.0254 ± 0.0003	0.0264 ± 0.0006	0.0255 ± 0.0002
stitched stage, $d = 7$	0.0235	0.0196 ± 0.0002	0.0198 ± 0.0003	0.0196 ± 0.0003
stitched stage, $d = 9$	0.0183	0.0153 ± 0.0004	0.0160 ± 0.0005	0.0154 ± 0.0005
stitched stage, $d = 11$	0.0142	0.0123 ± 0.0001	0.0129 ± 0.0001	0.0125 ± 0.0000
transformer, $d = 7$	0.0064	0.0050	0.379	0.029
transformer, $d = 11$	0.0019	0.0016	0.479	0.069
recurrent, $d = 5$	0.0286	0.0570	0.331	0.057
2023 device, $d = 5$	0.2562	0.2442	0.3902	—
BT device, $d = 3$	0.2566	0.2449	0.2463	—
BT device, $d = 5$	0.1074	0.1020	0.1156	—

D.3 Per-round rate, buffer and composed error

For $q \in [0, 1/2]$ and positive integer T, define the composition-equivalent rate $\varepsilon_{\text{eff}} \in [0, 1/2]$ as the unique solution of

$$q = \frac{1 - (1 - 2\varepsilon_{\text{eff}})^T}{2}, \qquad \varepsilon_{\text{eff}} = \frac{1 - (1 - 2q)^{1/T}}{2}.$$

The forward expression is the odd-parity probability of T independent, identically distributed Bernoulli flips. We use its inverse to normalise commit errors across T; this definition does not assume independent per-round errors. At fixed T the conversion preserves error rankings. Table 12 applies it to the per-commit errors of Table 5; Figure 8 sweeps the buffer given to correlated matching, and Figure 10 composes the decisions of Table 5 over blocks of B windows.

E Committing inside a device run

A device run holds one logical state for its whole round count, so its own earlier rounds are the predecessor of anything committed inside it. We cut a run into windows of W slices with the last window absorbing the remainder, commit at the end of each with zero buffer, and compose the window predictions with the ideal soft-XOR logit $-2 \operatorname{artanh}\left((-1)^n \prod_j \tanh(l_j/2)\right)$ for window logits $l_1, \ldots, l_n$. This is the same parity parameterisation as in Appendix F.1. The run's recorded

Table 12: **Composition-equivalent per-round error versus distance.** Per-commit errors q from Table 5 are converted as $\varepsilon_{\text{eff}} = [1 - (1 - 2q)^{1/T}]/2$, with $T = d + 1$. For learned models, the conversion is applied to the mean error over training runs. Rates and the relative change from $d = 5$ to $d = 9$ are computed from unrounded values and rounded only for display. **Bold** marks the lowest value within each column.

composition-equivalent rate	$d = 5$	$d = 7$	$d = 9$	d=5 to d=9
memory-trained, no buffer	0.05106	0.06679	0.09603	+88.1 %
memory-trained, one window of buffer	0.08898	0.12420	0.14333	+61.1 %
trained for it, no buffer	0.02323	0.02441	0.02638	+13.5 %
no buffer, correlated matching	0.03104	0.03349	0.03491	+12.5 %
one window of buffer, matching	0.02643	0.02736	0.02738	+3.6 %
one window of buffer, trained for it	**0.01495**	**0.01360**	**0.01332**	**−10.9 %**

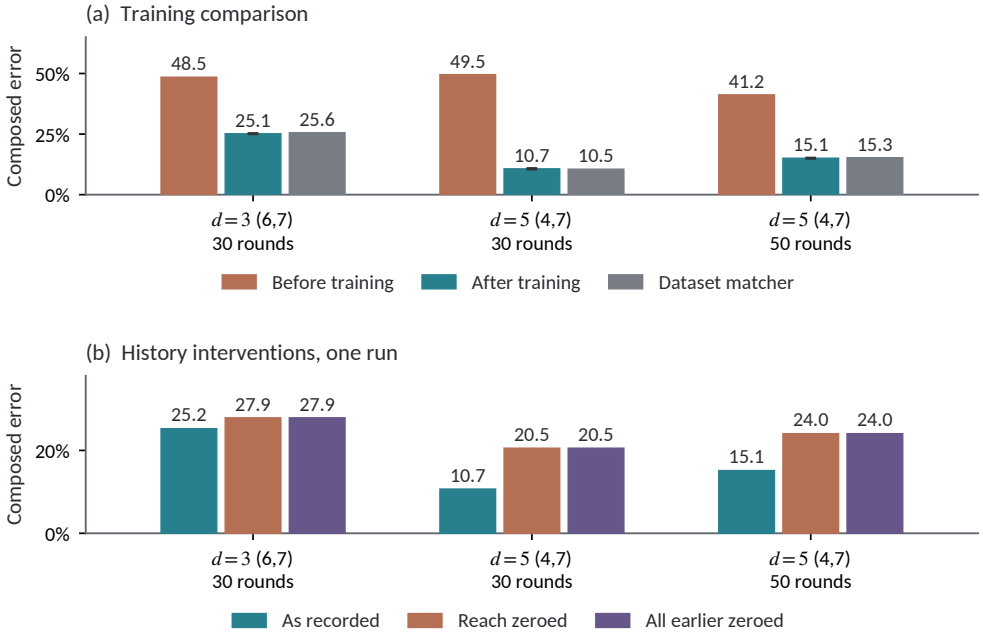


Figure 12: **Composed decoding on real device blocks.** (a) Composed error before and after fine-tuning, compared with dataset matching on identical shots. Before-training bars use one model. After-training bars show three-run means ± sample SD. (b) Errors from one trained checkpoint per device condition, with history recorded, zeroed within the receptive field, or zeroed at all earlier rounds. Columns identify the same device grid and round count in both panels. Parenthesized pairs identify the grids. Numbers above bars are errors in percent.

readout supervises that composition and is the only label used. On the below-threshold data the composed predictions reproduce the dataset's error suppression factor of 2.04 to within 0.2 percent. A commit is interior when its window opens at or beyond $R - 1$ slices, so that its receptive field clears the preparation. The thirty-round $d = 3$ run is cut 8, 8 and 14, the thirty-round $d = 5$ run 15 and 15, and the fifty-round $d = 5$ run 25 and 25. The procedure uses the fine-tuning stage's hyperparameters, with the run's own earlier windows supplying the predecessor. It uses the same stratified 50/25/25 split: validation blocks select the exponential moving average, and test blocks are scored once.

The three grid/round conditions have 2/1/1 interior commits and 8/12/12 predecessor rounds within reach, respectively.

Zeroing the receptive field. The interior commit's receptive field opens $R - 1 = 2L + 2$ slices before its window. Zeroing the run inside that span and recomputing that one commit, with every other window's decision taken from the run as recorded, gives the reach-history-zeroed series in Figure 12b. Zeroing every earlier slice gives the all-earlier-zeroed series. The two agree to the

Table 13: **Decoding whole device runs and composed interior commits.** The upper block scores each run as one decision. The lower block scores the composition of its window decisions. Comparisons within each block use identical shots and dataset matching predictions. After-stage entries are three-run means ± SD. BT denotes the below-threshold dataset. **Bold** marks the lowest error within each block and column.

Condition	2023, $d = 5$	BT, $d = 3$	BT, $d = 5$
dataset-provided correlated matching	0.25624	0.25664	0.10744
one commit over the run	**0.24424**	**0.24488**	**0.10200**
matching on the test blocks	0.25624	0.25552	0.15264
commits composed, before the stage	—	0.48496	0.41192
commits composed, after the stage	—	**0.25149 ± 0.00086**	**0.15061 ± 0.00072**

last bit at all three cuts, because what lies before the span never enters the commit. The nominal windows W are 8, 15 and 25 slices while $R - 1$ is 8, 12 and 12, so the distance a commit reads is the one the computation graph sets.

F THE CONDITIONAL FRAME–READOUT TASK

The circuit. Two rotated patches are stacked, patch A below and patch B above, separated by one row of data qubits. While the patches are apart those qubits are outside the code and the boundary stabilizers do not reach into them. To merge, the row is reset and the stabilizers of the combined patch are measured for $T = d + 1$ rounds. To split, the row is measured out and the two patches resume. The stabilizers of the merged patch that multiply to $Z_A Z_B$ are identified by solving over $\mathrm{GF}(2)$: eight, eighteen, thirty-two and fifty of them at $d = 3, 5, 7$ and 9, of which $d + 1$ exist only while merged. Stacking the patches side by side instead gives $X_A X_B$, which the committed frames do not correct. We therefore use the vertical arrangement to measure $Z_A Z_B$, whose outcome is corrected using the committed frames.

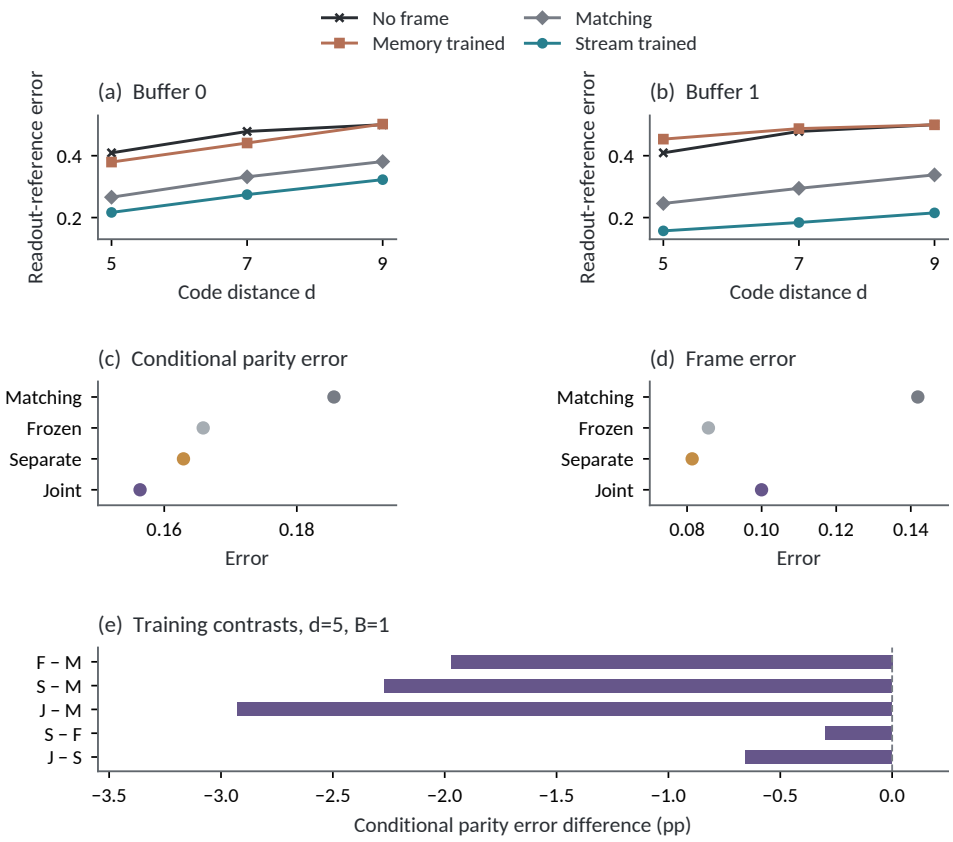
Construction checks. STIM's generator makes square patches only, so the layout, the four-step schedule and the boundary rule were rebuilt. A single patch built here has the same detector error model as STIM's `rotated_memory_z`, term for term, at $d = 3$ (219 terms, no differences, code distance 3). With no noise the solved product equals $Z_A Z_B$ on every shot at $d = 3, 5, 7$ and 9. Detectors are keyed on a stabilizer's support and compare only with the round before, since a boundary stabilizer changes support when the patches merge and again when they split. With that rule STIM accepts the circuit without gauge detectors.

Controls. Over 20,000 shots at $d = 3, 5, 7$ and 9: supplying the true frames and true readout correction recovers q_0 on every shot. Flipping the one consumed bit makes it wrong on every shot. Two perturbations whose parity cancels leave it right. The commit errors measured inside these runs reproduce every entry of Table 5 to within 0.008, at all three distances, both buffers and both decoders, and the window flip rates agree with its constant predictor.

The frame alone. In Figure 13, we supply each method with the merge's own readout error $r = m_{\mathrm{raw}} \oplus q_m$, so the arms differ in the frame and nothing else. Given the same readout the fine-tuning stage leads correlated matching at all six settings, by 15 to 19 percent with no buffer and by 36 to 38 percent with one window, the same order as the per-window margins of Table 5. At distances 5 and 7 with one window the standard procedure has a higher error than the no-frame reference, so correcting the conditional parity with it has higher error than applying no frame correction.

References and raw readout. Set $N = 17 + B$, $t_0 = (N - B)T$, $t_m = NT$ and $t_d = (N + 1)T$. Let $z_i(t)$ be the noiseless reference measurement of patch i's fixed representative on the noisy state. Then $q_0 = z_A(t_0) \oplus z_B(t_0)$, $q_m = z_A(t_m) \oplus z_B(t_m)$ and $f_i = z_i(t_m) \oplus z_i(t_0)$. If P is the selected old-stabilizer parity measured before merge and N_ℓ the new-stabilizer parity in merge round ℓ, scoring uses $m_{\mathrm{raw}} = P \oplus N_T$. At $d = 5$ the solved operator product uses 12 old and 6 new stabilizers. The old parity is not fixed to zero on noisy shots. Its signs are recoverable from prior detector parities in these initialized circuits.

Training uses $J_1 = P \oplus N_1 \oplus q_0$ and readout component $r_1 = P \oplus N_1 \oplus q_m$, whereas scoring uses $J = J_T$. Writing $D_\ell = N_\ell \oplus N_{\ell-1}$, the detector parity $K = \bigoplus_{\ell=2}^{T} D_\ell = N_T \oplus N_1$ follows because



F / S / J: frozen / separate / joint. M: joint matching.
40,000 shots. One operation model per condition.

Figure 13: **Readout and training-objective controls.** (a,b) Conditional parity error with reference readout and zero or one buffer window. Curves compare no frame, memory-trained frames, matching frames and stream-fine-tuned frames. Stream-frame points average three runs. (c–e) Conditional parity error, per-window frame error and conditional-error differences at $d = 5$, $B = 1$, for frozen (F), separate (S) and joint (J) training. Matching is joint in (c,e) and causal frame matching in (d). M denotes joint matching, and differences are in percentage points. All evaluations use $p = 0.007$ and 40,000 shots per condition, shared across methods, with one plotted operation model per condition. All operation errors are on J relative to q_0.

every intermediate N_ℓ appears twice. Thus $J_T = J_1 \oplus K$, and inference adds the same K to the predicted bit before scoring. For the label identity, direct substitution gives

$$f_A \oplus f_B = q_m \oplus q_0, \tag{6}$$

$$r \oplus f_A \oplus f_B = (m_{\text{raw}} \oplus q_m) \oplus (q_m \oplus q_0) = m_{\text{raw}} \oplus q_0 = J. \tag{7}$$

Consequently $(m_{\text{raw}} \oplus \hat{J}) \oplus q_0 = J \oplus \hat{J}$, which vanishes exactly for a correct conditional prediction. Oracle readout alone leaves error $f_A \oplus f_B$, explaining the nonzero no-frame curves. The scored reference q_0 is the noisy pre-block value. Relative to an ideal input reference $q_\star$, the incoming correction $C_0 = q_0 \oplus q_\star$ combines with the block correction as $E = J \oplus C_0$.

The outcome is consumed at the last merged round. Every decoder sees the detectors through that round. Truncating the detector error model at this deadline retains each mechanism's earlier detectors, producing the open edge described in Appendix C. Joint matching is PyMatching on that cut detector error model with J as its observable and correlations enabled, the whole stream of $17 + B$ windows as its history. For joint matching without correlations, including the T rounds after the merge reduces error by about 0.002 at each distance. At the merge deadline, enabling correlations reduces error by 0.014, 0.023 and 0.022 at distances 5, 7 and 9, respectively. The pipelines commit each patch's frame with the one-window decoders of Table 5, hard bits, four windows of history and

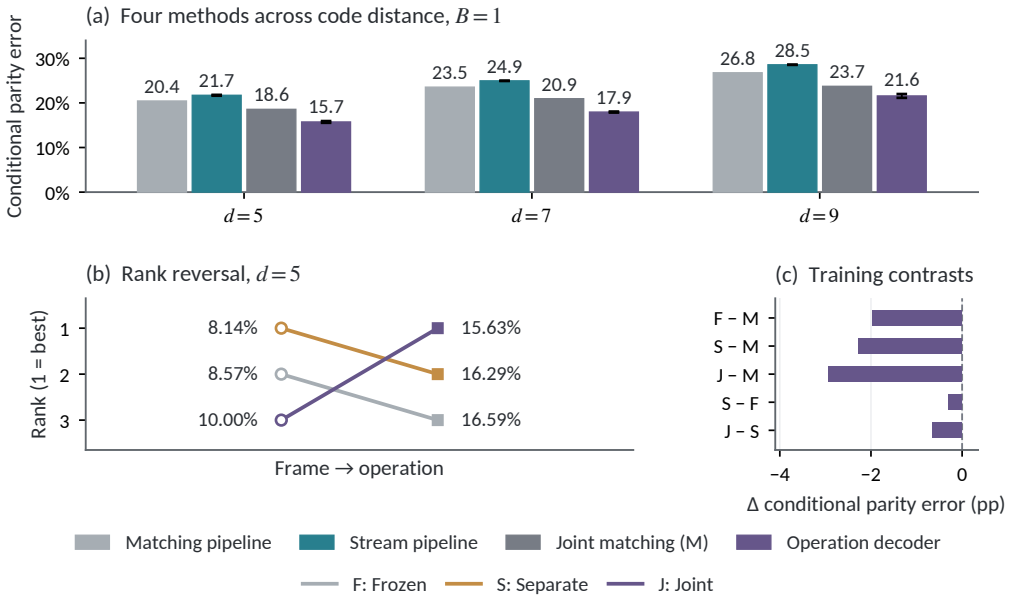


Figure 14: **Conditional frame–readout decoding under different training objectives.** (a) Conditional parity error at $B = 1$ for two frame pipelines with matching readout, joint matching, and the operation decoder. The stream pipeline and operation decoder show means $\pm$ sample SD ($n = 3$). (b,c) Frame-to-operation ranks and operation-error differences, respectively, at $d = 5$, $B = 1$. F, S and J denote frozen frames, separate labels and joint training. M denotes joint matching. Differences are in percentage points. Evaluations use $p = 0.007$ and 40,000 shared shots per condition. Panels (b,c) use one matched training run per objective. All operation-error panels score J, relative to q_0.

the merge as buffer, and decode the merge readout with the same matcher built for that observable alone. The readout network reads the $B + 1$ windows before the merge plus the merge itself. The joint matcher reads more history than any learned part.

The operation decoder. The frame decoder is the one-window checkpoint of Table 5, committing each of the B windows the operation consumes as before, and it is trained further here. The readout network is the convolutional trunk of Appendix G, nine layers of width 128 at every distance, on the merged lattice of both patches and the gap, from random weights, with a head that produces logits at occupied sites in the merged rounds and pools them through a soft exclusive-or, a parity structure the frame decoder's mean pool does not have. The $2B$ frame logits and the readout logit are combined by the same soft exclusive-or. We train the combined output on J_1 with binary cross-entropy and update both networks jointly. We select one training configuration on a held-out development set before the reported runs and use it at every distance and schedule: AdamW at 10^{-3} for the readout network and 2×10^{-4} for the frame decoder, weight decay 3×10^{-3}, 1,000 warm-up steps, a cosine schedule to a tenth of the peak, bfloat16 autocast for the readout trunk, and gradient clipping at 1. We use the EMA of Appendix G and a low-noise opening at $p/10$ for the first 5 percent of steps, annealed to p by 20 percent. Each training run uses 40,000 steps with batches of 512 samples. Repetition settings for the distance sweep, longer blocks, and objective controls are given in Appendix F.1. Final EMA weights are evaluated on the primary test set and an independent replication set, each with 40,000 shots. Appendix F.1 reports the aggregated distance sweep. In the single-run component controls, the frames accompanying the operation decoder are trained for the operation: their per-window error over the consumed windows is 0.100, 0.116 and 0.147 at distance 5 for $B = 1, 2, 3$ against 0.086, 0.125 and 0.159 for the frozen checkpoint, and 0.109 against 0.102 at distance 7. At $B = 2$, allowing both networks to adapt improves the conditional score: with the frame decoder frozen the system has error 0.2288 against joint matching's 0.2267 at distance 5 on identical shots, and joint training takes it to 0.2067 on J. Its cost is paid once per operation, on the merge's rounds, in addition to the frame decoder's per-round cost.

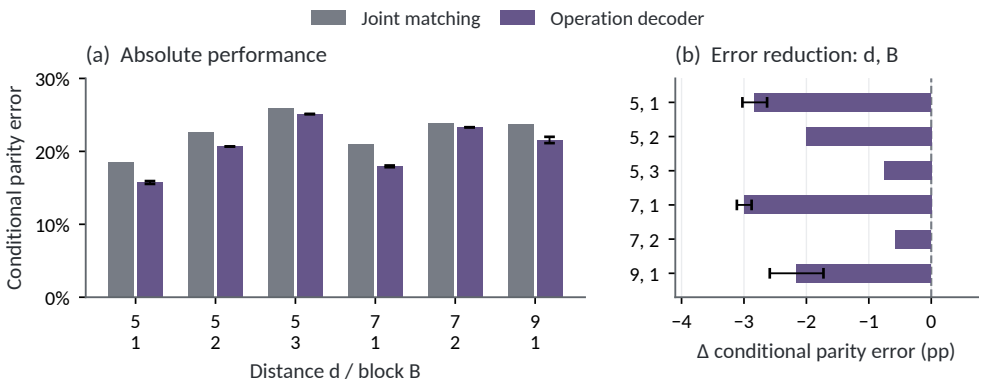


Figure 15: **Conditional decoding across block lengths.** (a) Conditional parity error for the operation decoder and joint matching across distance d and block length B. (b) Their error difference, operation decoder minus matching, in percentage points. Each condition uses $p = 0.007$ and 40,000 shared shots. At $B = 1$, bars show three-run means with sample SD. Longer blocks use one training run per condition. Difference-panel error bars show training-run SD.

F.1 Joint supervision and matched controls

For scoring, let a_i and $\hat{a}_i$ denote the true and predicted window corrections followed by r_T and $\hat{r}_T = \hat{r}_1 \oplus K$. Let $e_i = \hat{a}_i \oplus a_i$. For the hard XOR prediction $\hat{J} = \bigoplus_i \hat{a}_i$, XORing the predicted components and then the true components gives $\hat{J} \oplus J = \bigoplus_i (\hat{a}_i \oplus a_i) = \bigoplus_i e_i$ by associativity and commutativity. Thus, for $n = 2B + 1$ components and $\mathbf{e} = (e_1, \ldots, e_n)$,

$$\Pr(\hat{J} \neq J) = \Pr\left(\bigoplus_{i=1}^{n} e_i = 1\right) = \sum_{\substack{\eta \in \{0,1\}^n \\ \|\eta\|_1 \text{ odd}}} \Pr(\mathbf{e} = \eta).$$

This identity requires no independence assumption. For $B = 1$ there are three components, so

$$\Pr(\hat{J} \neq J) = \Pr(100) + \Pr(010) + \Pr(001) + \Pr(111), \tag{8}$$

where each event lists (e_A, e_B, e_r). The other four patterns have even parity and are correct. Marginal frame errors sum over different combinations of these events. Their joint frequencies determine the output error, motivating supervision of the combined prediction.

For example, let two error bits have probabilities $\Pr(00) = 0.8$ and $\Pr(10) = \Pr(01) = 0.1$. Both marginal errors are 0.1, but the XOR error is 0.2. A second system with $\Pr(00) = 0.85$ and $\Pr(11) = 0.15$ has larger marginal errors, 0.15 each, and zero XOR error. All unlisted patterns have zero probability. A third error bit fixed at zero embeds this example in the three-component setting.

For logits $\ell_1, \ldots, \ell_{2B+1}$, the differentiable parity head is $q = [1 - \prod_i (1 - 2\sigma(\ell_i))]/2$, trained with $\mathcal{L}_J = \mathrm{BCE}(q, J_1)$ in the ideal parameterisation. For binary component values a_i, $1 - 2a_i = (-1)^{a_i}$, so their product is $(-1)^{\sum_i a_i}$ and this formula equals their XOR. During training it supplies a continuous parameterization of that parity output. If $p_i = \sigma(\ell_i)$ are interpreted as Bernoulli marginals conditional on common information x, conditional mutual independence is sufficient for q to equal the conditional XOR probability. We train q directly against J_1, without assuming independence or requiring calibrated component probabilities.

In ideal arithmetic, when every $\ell_i \neq 0$, the sign of $1 - 2\sigma(\ell_i)$ gives $\mathbf{1}[q > 1/2] = \bigoplus_i \mathbf{1}[\ell_i > 0]$. The implementation evaluates a stabilised parity logit $\tilde{\ell}_\oplus$ and uses BCE with logits. It computes the product magnitude from $\sum_i \log \max(|\tanh(\ell_i/2)|, 10^{-30})$, assigns its sign using the parity of $\mathbf{1}[\ell_i > 0]$, clips the signed product to $[-1 + 10^{-6}, 1 - 10^{-6}]$, and returns $\log(1 - z) - \log(1 + z)$ for that clipped product z. Deployment reports $\hat{J}_1 = \mathbf{1}[\tilde{\ell}_\oplus > 0]$ and $\hat{J}_T = \hat{J}_1 \oplus K$. Since both the label and prediction receive K, $\hat{J}_T \oplus J_T = \hat{J}_1 \oplus J_1$. At zero component logits or numerical

underflow, the stabilised threshold need not equal thresholding the ideal q. The floor and clipping operations are also part of the training computation. For the three matched joint/separate pairs at $d = 5$, $B = 1$, the archived primary and replication evaluations contain no zero component logits and satisfy $\hat{J} \oplus J = \bigoplus_i e_i$ for all 480,000 predictions from the six models evaluated on the same 80,000 shots.

The separate-supervision control uses $\mathcal{L}_S = (2B+1)^{-1} \sum_i \mathrm{BCEWithLogits}(\ell_i, a_i^{(1)})$, where $a_i^{(1)}$ are the $2B$ window increments followed by r_1. Each loss uses its native gradient scale under the shared optimizer settings below.

At $d = 5$, $B = 1$, the two controls use the same networks, input spans, the same initialization recipe, optimizer hyperparameters, and 40,000-step budget of 512 trajectories per step. Joint supervision yields lower conditional error in all three matched training runs on both test sets (the matched-objective table below). The frozen-frame control also uses $\mathcal{L}_J$, but updates only the readout network. Its comparison with joint training measures the effect of allowing the frame network to adapt.

Experimental repetitions. At $B = 1$, each distance uses three independent operation-training runs with different random initializations of the readout network, independently sampled training data, and the corresponding pre-trained frame checkpoints. We report the arithmetic mean and sample standard deviation across runs. The separate-supervision control at $d = 5$, $B = 1$ uses the same three matched initializations, frame checkpoints, and training-data randomization as joint supervision. Longer-block conditions use one training run per condition. Each fitted model is evaluated on the same primary test trajectories and on an independent replication set. Each set contains 40,000 shots at $p = 0.007$.

d	Test set	Ours (%, mean ± SD)	Matching (%)	Reduction
5	Primary	15.732 ± 0.198	18.560	15.23%
5	Replication	15.444 ± 0.350	18.607	17.00%
7	Primary	17.947 ± 0.118	20.942	14.31%
7	Replication	17.961 ± 0.224	21.002	14.48%
9	Primary	21.563 ± 0.430	23.720	9.09%
9	Replication	21.914 ± 0.361	23.817	7.99%

The mean conditional error is below matching at all three distances on both test sets. The mean margin is smaller at $d = 9$.

Matched training objectives at $d = 5$			
Test set	Joint (%)	Separate (%)	Reduction (pp)
Primary	15.73 ± 0.20	16.32 ± 0.12	0.59
Replication	15.44 ± 0.35	15.97 ± 0.05	0.53

Reduction is separate minus joint, in percentage points, calculated before rounding.

G ARCHITECTURES, PROCEDURES, AND THE CHECKPOINT MAP

G.1 KEEPING PACE WITH THE SYNDROME

Table 14: **Streaming-kernel latency and accuracy.** Scheduled arrivals span 64 trajectories per distance. Times are in microseconds. Response is the largest observed delay from a window's final input round to its decision. Error columns compare the streaming and offline evaluations of the same weights, followed by the kernel with standard-procedure weights.

	arrival	largest observed	per-commit error		
	period	response	the kernel	offline	standard
$d = 5$	6.00	59.4	0.1240	0.1240	0.2633
$d = 7$	8.00	118.2	0.1624	0.1626	0.3333

Rate and latency. A deployable advantage must keep pace with the syndrome. The stream-fine-tuned decoder runs inside a persistent kernel on one NVIDIA H100 that carries the queues of con-

volution inputs across window boundaries, as defined in Appendix A, driven by scheduled arrivals at one round per microsecond. The kernel sustains the scheduled arrival rate over more than half a million windows, with decision spacing at the arrival period. The buffer is what a decision may read, which Section 3.1 sets. The response time is what elapses from a window's last round to its decision. Its largest observed value is 59.4 microseconds at distance 5 and 118.2 microseconds at distance 7 (Table 14). Operations consuming a frame wait for its response. This separates sustained syndrome throughput from the latency of dependent logical operations (Skoric et al., 2023). At distance 5 the response stays below the latency the matching decoder on this machine reports (Google Quantum AI and Collaborators, 2025). Accuracy through the kernel matches the same weights scored offline to within 0.0002 (Table 14), and the audited commits match an exact replay.

The kernel. The deployment of Section G.1 runs the stream-fine-tuned checkpoint inside one persistent kernel with an fp16 multiply datapath, weights held on chip in fp16 at distance 5 and in e4m3 at distance 7, on 16 clusters of 6 blocks at distance 5 and 15 clusters of 8 blocks at distance 7, which is 96 and 120 of the 132 streaming multiprocessors of one NVIDIA H100 80GB. The kernel keeps the weights and the per-round working set on chip and carries the causal cut of Definition 4 across window boundaries. Decisions are checked against an exact replay reference in the kernel's own precision on 4,096 audited commits per weight set.

G.2 The decoder and how it is trained

The causal stack. The decoder of Definition 2 takes the detector volume (B, R, E, E), where E is the lattice side and R the round count, and returns one logit per commit. An embedding maps each detector site to H channels. $L = d$ residual blocks follow, each applying a pointwise map, a width-3 causal temporal convolution at unit dilation, and a second pointwise map, added to the block input. A fixed-statistic norm and SiLU precede a width-3 causal scatter, and the head averages over the T-round window and applies a linear map to one logit. Table 15 gives the per-layer shapes and Table 16 the sizes.

Table 15: **Per-layer shapes of the causal stack.** B denotes batch size, R input rounds, E lattice side length, and H channel width. Temporal convolutions have width 3 and unit dilation.

Stage	Layer	Input $\to$ Output
Embedding	pointwise	$(B, R, E, E) \to (B, R, E, E, H)$
Block $\times L$	pointwise in	$H \to H$
	causal conv, width 3	$H \to H$
	pointwise out	$H \to H$
	residual add	$H \to H$
Tail	norm, SiLU	$H \to H$
	causal scatter, width 3	$H \to H$
Head	window mean	$(B, T, E, E, H) \to (B, H)$
	linear	$H \to 1$

Table 16: **Causal-stack configurations by code distance.** L denotes residual-block count and H channel width. Parameter counts cover the full model.

	$d = 5$	$d = 7$	$d = 9$	$d = 11$
blocks L	5	7	9	11
width H	64	128	192	192
parameters	165,121	729,857	1,801,729	1,964,161

Recipes. Table 17 lists the three training stages. Every stage draws its circuits on the fly. The loss is taken against the committed observable flip, and the averaging decay is ramped to its value over the opening steps of memory training's stage. All three stages use the final EMA weights.

Table 17: **Training settings for memory, stitched and stream-fine-tuning stages.** Columns specify memory training, stitched training, and no-buffer fine-tuning. Buffered fine-tuning starts from stitched weights and adds lookahead as specified in Appendix B.2. EMA denotes exponential moving average.

Hyperparameter	Standard procedure	Stitched	Repair
Pre-trained Init	–	standard procedure	standard procedure
Data	on the fly	on the fly	on the fly
Training Samples	2.5×10^8	2.0×10^7	1.0×10^7
Optimizer	AdamW	AdamW	AdamW
Learning Rate	10^{-3}	2×10^{-4}	2×10^{-4}
LR Schedule	Cosine	Cosine	Constant
Warmup Steps	1,000	1,000	–
Minimum LR	$0.1\times$ peak	$0.1\times$ peak	–
Weight Decay	3×10^{-3}	3×10^{-3}	10^{-2}
Gradient Clipping	1.0	–	–
Batch Size	1,024	512	256
Steps	240,000	40,000	40,000
EMA Decay	0.9998	0.9998	0.9998
Precision	bfloat16	FP32	FP32
Loss Function	BCEWithLogitsLoss	BCEWithLogitsLoss	BCEWithLogitsLoss
Training p	0.007	0.007	0.007
Predecessor Rounds	0	two experiments	$3T$

G.3 Transfer and the checkpoint map

The transfer setting. Both pre-trained decoders are used with their authors' weights and code, and each added stage runs at that model's own training noise level and at batch 64, with the same 40,000 steps, target and schedule as the convolutional stack's. Scoring maps the stream's detectors into the model's own coordinate convention. The transformer's carried entries in Appendix D.2 score its whole-experiment output against the second experiment's readout. Table 18 gives both of its readings of the stream commits. For the transformer the stage is at $p = 0.005$, its own training noise level, and at the convolutional stack's learning rate. The transformer column of Table 2 then scores that checkpoint on the $p = 0.007$ stream the rest of the paper uses, so it crosses noise levels as well as protocols. For the graph decoder it is $p = 0.005$ at the checkpoint's own learning rate, 10^{-4}: the graph family publishes its optimizer setting with its weights, and the four published distances share one setting. Two properties of that family fix the rest of the protocol. Its input is a set of triggered detectors rather than a fixed tensor, so the number of input slices is a choice. It is made on a development stream disjoint from the frozen corpus, by minimizing the *pre-trained* decoder's own per-window error over all five public checkpoints at each distance and over one- and two-window spans ending at the commit. The selected span is one window at all three distances. Selected checkpoint d_t values are 5, 11 and 11 at $d = 5, 7, 9$, respectively. Fine-tuning inherits these choices. Its pre-trained weights are trained at p between 0.001 and 0.005, so the stream corpora for this comparison are generated at 0.005 by the same frozen generator, and every row in the graph columns of Table 2 decodes the identical commits. The adapter is held to the checkpoint's own published setting before any stream number is read: the checkpoint each column uses decodes its own reset-bounded memory experiment below the matching baseline at every distance (Table 19). The fine-tuning stage uses three independent training runs per distance, whose standard deviations fall in the fourth decimal.

Transformer supervision and output semantics. Let $g_\theta(x)$ be the transformer's single output logit, and let x_{4T} and x_{3T} share a start. For the released cumulative-output model, the inference adapter is $\mathbf{1}[g_\theta(x_{4T}) \geq 0] \oplus \mathbf{1}[g_\theta(x_{3T}) \geq 0]$. Fine-tuning instead minimizes

$$\mathcal{L}_{\text{local}}(\theta) = \text{BCEWithLogits}\big(g_\theta(x_{4T}), z_4 \oplus z_3\big). \tag{9}$$

The loss supervises the direct $4T$ output as the window prediction. Table 2 compares the released cumulative-prefix adapter with this locally supervised head. Table 18 additionally reports both output conventions for each checkpoint. Training uses $p = 0.005$. The principal comparison at $p = 0.007$ tests noise transfer alongside local supervision. The final column evaluates the direct

output at the training noise, scoring the released cumulative head and the adapted local head against the window label.

A separate stitched check at $p = 0.005$ also uses the released cumulative convention. On 20,000 shots, differencing the stitched prediction with the predecessor prediction gives errors 0.20070 and 0.33035 at $d = 7$ and 11, respectively, versus isolated errors 0.00465 and 0.00130 on those shots. This adapter also yields higher error than isolated evaluation. The direct carried columns test the cumulative head against the second experiment's label, combining predecessor context with a change in scoring convention.

Table 18: **Two transformer readings of stream commits at $d = 7$.** Direct scores the $4T$ output. Differenced XORs the $4T$ and $3T$ decisions. Released weights predict cumulative parity. Local fine-tuning supervises the direct window prediction. The first two columns use $p = 0.007$. The last uses the model's training noise level, $p = 0.005$. **Bold** marks the lowest error within each column.

	$p = 0.007$		the model's own
per-commit error	direct	differenced	direct, $p = 0.005$
causal correlated matching	0.2128		0.1473
pre-trained weights	0.4616	0.4094	0.4618
after the fine-tuning stage	**0.1734**	**0.1735**	**0.1039**

Table 19: **Reset-bounded benchmark of the pre-trained graph decoder.** Logical error per memory experiment at $p = 0.005$, using 100,000 shots per column and the checkpoints evaluated in Table 2. **Bold** marks the lowest error within each column.

memory experiment, $p = 0.005$	$d = 5$	$d = 7$	$d = 9$
correlated matching	0.01406	0.01673	0.00778
pre-trained graph decoder	**0.00908**	**0.01241**	**0.00640**

Table 20: **Checkpoints underlying the reported results.** Rows link training procedures to their figures and tables. Pre-trained transformer, graph and recurrent weights come from Yan et al. (2026b), Lange et al. (2025) and Varbanov et al. (2025), respectively. Device data come from Google Quantum AI (2023; 2024).

Name in the paper	Training	Numbers it produces
memory-trained	standard procedure	Table 5 row 2, Figure 11a, Figure 2
stream-fine-tuned, three runs	stream fine-tuning	Table 5 row 3, Figure 2
one stage	fine-tuning from reset	0.1240 and 0.1626, kernel rows
stitched, three runs	stitched stage	stress-test rows of Appendix D.2
further-training control	memory training again	Figure 6
matched-history	fine-tuning at 0–3 windows	Figure 7
fill-ratio training	stitched at h rounds	Figure 7
pre-trained transformer	authors' weights	Table 2, Table 18
transformer, one stage	stream fine-tuning	Table 2, Table 18
pre-trained graph decoder	authors' weights	0.2062, 0.3027, 0.3777, Table 2
graph decoder, one stage	stream fine-tuning	0.0909, 0.1296, 0.1673, Table 2
recurrent	authors' package	Figure 11b
device checkpoints	fine-tune on device data	Table 13 upper block, Figure 11c
device, committed in run	the same, composed	Table 13 lower block, Appendix E