Sparse Streaming Neural Decoders for Real-Time Quantum Error Correction

Eric Jin[1] **Susanne F. Yelin**[2] **Andi Gu**[2]
[1]University of California, Berkeley [2]Department of Physics, Harvard University

Abstract

Real-time classical decoding is a central bottleneck for fault-tolerant quantum computing: for superconducting qubits, a decoder must process each syndrome round within roughly a microsecond and at bounded latency, or an exponentially growing backlog renders error correction infeasible. Neural-network decoders have reached the highest accuracies and can now run in real time at intermediate code sizes, but whether they can be deployed at scale remains unproven. We co-design a sparse neural decoder with its hardware, mapping the network onto a spatial-dataflow FPGA as a fully unrolled, data-independent pipeline; a complete distance-11 surface-code decoder fits on a single FPGA and sustains the microsecond-per-round budget at deterministic latency. The same design extends to high-rate quantum LDPC codes such as the $[[144, 12, 12]]$ Gross code, and because its heavy computation is feed-forward rather than iterative, it distributes across devices to reach larger codes without sacrificing throughput.

1 Introduction

Quantum error correction (QEC) protects a computation by repeatedly measuring parity checks on the physical qubits. Each round yields a *syndrome*, from which a classical *decoder* must infer the errors in real time. Three quantities characterize a decoder: accuracy, throughput, and latency. Throughput is a hard constraint: the decoder must consume syndromes at least as fast as they are produced—roughly one round per microsecond for superconducting qubits—or its backlog grows without bound Terhal [2015]. In contrast, accuracy and latency have no fixed target; the goal is simply to improve each. Higher accuracy lowers the logical error rate, and thus the number of physical qubits needed for a target reliability. Lower latency is important because the decoder lies on the critical path: under time-optimal scheduling, the decoder's reaction time, rather than the quantum hardware, sets the algorithm's execution time Fowler [2013], Yang et al. [2026], so a decoder $k\times$ faster yields a computation $k\times$ faster.

Two families of decoder approach this speed requirement from opposite sides. Fast algorithmic decoders—matching, union-find, and belief propagation—already run in real time on FPGAs and ASICs Liyanage et al. [2024], Barber et al. [2025], Maurer et al. [2025], but at reduced accuracy. Neural-network decoders reach the highest accuracies Gu et al. [2026], Senior et al. [2025] and, in recent work, sub-microsecond throughput at intermediate code sizes Senior et al. [2025], yet they run on commercial accelerators, where latency remains below $500\ \mu$s Senior et al. [2025] and, unlike throughput, directly slows a reaction-time-limited computation. The open problem is to deliver neural accuracy at deterministic low latency, at larger code distances, and for high-rate quantum LDPC codes such as the bivariate-bicyclic "Gross" codes Bravyi et al. [2024], which promise the most qubit-efficient route to fault tolerance. This points to reconfigurable hardware placed alongside the controller: an FPGA delivers deterministic, low latency with no host round-trip, and specialized hardware, low-precision arithmetic, and windowing are the natural means to these ends Senior et al. [2025], which we pursue here.

Preprint.

Our starting point is the convolutional decoder of Gu et al. [2026], whose local, translation-equivariant structure attains state-of-the-art accuracy on both surface and BB codes and is especially well matched to spatial-dataflow hardware. Building on it, this work makes three contributions:

1. A **hierarchical sparse decoder** (Section 2): a convolutional spatial backbone feeding a recurrent outer decoder that predicts logical flips in a single pass. It couples the problem's inherent spatial locality with co-designed sparsity in the channel dimension and low-precision arithmetic to reach an FPGA-viable compute budget.
2. A **measured FPGA implementation** (Section 3): a fully unrolled spatial-dataflow pipeline that places a complete distance-11 surface-code decoder on a single AMD Versal HBM FPGA (about half its LUTs, a third of its DSPs) at deterministic, data-independent latency, and scales to larger codes across devices by pipeline parallelism.
3. An **extension to high-rate qLDPC codes** (Section 3): the dataflow must read each check's neighbors in one cycle, which the irregular BB Tanner graph appears to preclude. We prove the IBM BB family admits an *affine* activation-banking scheme that serves every check's neighbors in a single cycle, making BB codes single-chip-mappable; the $[[144, 12, 12]]$ Gross code (144 data qubits, 12 logical qubits, distance 12) is the target for a head-to-head with the Relay-BP FPGA decoder Maurer et al. [2025].

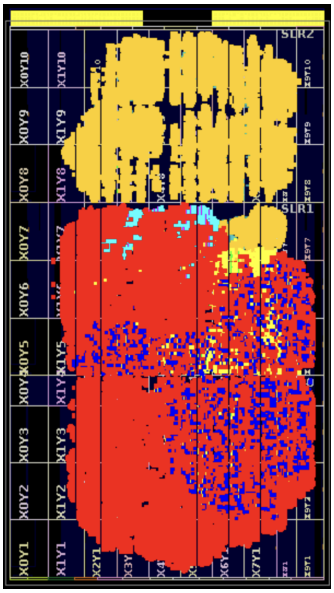
2 Decoder architecture

The decoder inherits the convolutional backbone of Gu et al. [2026], whose structure follows the geometric regularity of QEC codes. Under repeated syndrome measurement, stabilizers form a regular spacetime lattice in which every site has the same local neighborhood up to translation. Two properties follow. *Locality*: errors produce spatially localized syndromes that local processing can progressively resolve. *Translation equivariance*: identical structure at every site means the same rules apply everywhere. A stack of L convolutional layers realizes both, each performing learned, direction-dependent message passing between neighboring sites—a 3D convolution over the spacetime lattice for surface codes, a generalized convolution on the torus for BB codes—with weights shared across positions. Successive layers expand the receptive field, so after $L \sim d$ layers the network integrates information across the full code distance. A lightweight recurrent decoder—a single, ungated recurrent cell at the backbone's channel width—then consumes the backbone's per-round output and tracks the logical state as syndromes stream in. A third, purely geometric property is that the X- and Z-type checks sit on two sublattices related by a relabeling of the code, so the same convolutional weights serve both up to that relabeling—a symmetry the hardware later exploits directly (Section 3).

The accuracy-optimal backbone is dense and over-budget for an FPGA, so we make two co-design choices, distinct from the inherent properties above. First, we sparsify the channel dimension. A *depthwise* convolution keeps channels separate, $y_c(\mathbf{x}) = \sum_\delta w_{c\delta}\, h_c(\mathbf{x}{+}\delta)$, costing $O(n)$ multiplies per site rather than the $O(n^2)$ of a dense convolution that also sums over the n channels; the cross-channel mixing is then delegated to a *Monarch* layer Dao et al. [2022], which factorizes the dense mixing matrix $W \in \mathbb{R}^{n\times n}$ into two block-diagonal matrices and a permutation, $W = B_1\, P\, B_2$ (each B_i has $\sqrt{n}$ blocks of size $\sqrt{n} \times \sqrt{n}$), at $O(n\sqrt{n})$. Second, we quantize the convolutional backbone with quantization-aware training: INT8 activations and depthwise weights, with the Monarch weights reduced further to INT4; the outer RNN is not yet quantized, and Section 3 reports the resulting cost. Because the weights are fixed at inference, they are compiled into the datapath rather than fetched, so lower Monarch precision directly reduces the on-chip arithmetic.

3 FPGA implementation

We map the resulting network to an AMD Versal HBM FPGA (`xcv80`) as a *fully unrolled spatial-dataflow pipeline*: every layer is dedicated hardware and spatial positions stream through at one position per clock. Syndromes arrive as one $(d{+}1)^2$ grid per round—a distance-d surface code uses $O(d^2)$ physical qubits arranged in a $d \times d$ patch—and each layer emits round t once round $t{+}1$ arrives, adding fixed latency but never reducing throughput. Control flow is data-independent, so the pipeline consumes one round every $(d{+}1)^2$ cycles—$\sim$0.3 μs at $d{=}11$ and the target 500 MHz clock, within the 1 μs budget at any depth.



Block	LUT	FF	DSP58
RNN (`u_rnn`)	932,135 (36.2%)	128,466 (2.5%)	1,809 (16.7%)
Conv stack (8×)	345,751 (13.4%)	221,383 (4.3%)	2,256 (20.8%)
Dequant (`u_deq`)	4,329 (0.17%)	1,025 (0.02%)	32 (0.30%)
Pool (`u_pool`)	8,995 (0.35%)	799 (0.02%)	0
Readout (`u_ro`)	12,201 (0.47%)	1,679 (0.03%)	21 (0.19%)
Total	1,303,447 (50.6%)	353,372 (6.9%)	4,118 (38.0%)

Submodule	LUT	FF	DSP58
Monarch (`u_mono`)	23,815 (0.93%)	8,180 (0.16%)	0
Depthwise (`u_dw`)	15,563 (0.60%)	12,563 (0.24%)	250 (2.3%)
Skip FIFO	1,536 (0.06%)	256	0
Window assemble	913 (0.04%)	3,031 (0.06%)	0
Residual add	729 (0.03%)	769 (0.01%)	32 (0.30%)
Temporal buffer	∼300–1,100	∼60–200	0
Conv layer total	∼41,300 (1.6%)	∼24,600 (0.48%)	282 (2.6%)

Figure 1: **FPGA implementation of the complete** d=11 **decoder** on an AMD Versal HBM `xcv80` (2.57M LUT / 5.15M FF / 10,848 DSP58), synthesized with Vivado 2025.2 (RTL, out-of-context); II= 1 throughout, and table values are percentages of the device. *Left:* floorplan colored by block—the fp32 RNN tail dominates the die, and the design spreads across all three super-logic regions. Colors: orange, conv stack; red, RNN; blue, dequantization (INT8 → FP32); cyan, readout; yellow, average pooling; magenta, embedding; green, control. *Right, top:* per-module utilization of the full decoder (L=8, width 32). *Right, bottom:* breakdown of one convolution layer, split between the multiplierless Monarch (all LUT) and the DSP-packed depthwise. BRAM usage is entirely within convolution stack's temporal round buffers at 32 BRAM36 total, 0.86% of the device

Hybrid arithmetic. The two layer types map to complementary primitives. Depthwise convolutions use **DSP58 blocks**: each channel's 27-tap ($3 \times 3 \times 3$) neighborhood splits into nine groups of three taps, one DSP58 per group packing three INT8 multiply-accumulates, and the nine partial sums are reduced entirely within the DSP column by chaining the DSP58s along their dedicated cascade path. A parity bit tagging each site's sublattice drives a 2:1 mux on the DSP's constant-coefficient input, selecting the X- or Z-check weights ahead of the multiply with no added pipeline stage; the two sublattices thus share one physical multiplier at 1× DSP cost, rather than the naive implementation, which would cost 2×. Monarch pointwise layers instead use **distributed-arithmetic shift-add trees** (da4ml Sun et al. [2026]): since the Monarch weights are compile-time constants, each dot product against them reduces to a fixed pattern of shifts and adds rather than a multiply, and da4ml applies common-subexpression elimination across the many such patterns in the two Monarch factors to share adders and cut LUTs further. This hybrid outperforms either pure approach: a pure-LUT depthwise would need a separate shift-add circuit per sublattice, while DSPs are wasted on the small Monarch blocks.

Synthesis results. Figure 1 reports the measured utilization of the complete d=11 decoder: it fits on a single `xcv80`, drawing 50.6% of its LUTs and 38.0% of its DSP58 at II= 1. The eight-layer convolution stack (13.4% LUT, 20.8% DSP) and the fp32 outer RNN (36.2% LUT) dominate; the RNN's fabric fp32 arithmetic will shrink once mapped to the Versal hardened floating-point (DSP58) operators. A single convolution layer costs only ∼1.6% LUT and 2.6% DSP, split between the multiplierless Monarch pointwise (all LUT, no DSP) and the DSP-packed depthwise. Weights are constants baked into the circuit; only syndromes, activations, and a few rounds of temporal context reside on-chip.

Extending to high-rate qLDPC codes. The surface code is planar, so the dataflow reads each check's neighbors from a line-buffered window. Bivariate-bicyclic (BB) codes Bravyi et al. [2024] are higher-rate but non-planar: each check reads six data neighbors scattered across the activation

frame, which one memory cannot serve in a single cycle. We solve this at compile time: for the IBM BB code family, the frame admits an *affine* 6-bank labeling $b(x, y, \ell) = \ell + 2\,((x{+}y) \bmod 3)$ placing the six neighbors of every check in six distinct banks, so all reads complete in one cycle with constant, position-independent wiring. This holds for all seven codes of Bravyi et al. [2024] and maps to distributed RAM, keeping the convolution and da4ml layers identical to the surface-code chain. BB codes such as the $[[144, 12, 12]]$ Gross code thus become single-chip-mappable; the Gross code is our target for a head-to-head with the Relay-BP FPGA decoder Maurer et al. [2025].

Because the bank assignment depends only on the code's polynomial supports A, B, a compile-time script evaluates $b(x, y, \ell)$ once per code to produce the bank-labeling table and emits it as fixed circuit wiring in RTL, similar to the same generation step that produces the da4ml adder graphs and the Monarch permutation elsewhere in the design, after which Vivado synthesizes this wiring exactly as it does every other module. Extending to a different member of the BB family therefore requires no hand redesign of the window-assembly hardware: only re-running this script on the new polynomials and re-synthesizing the resulting new circuit hardware.

Scaling by pipeline parallelism. A single chip is not the ceiling: beyond its capacity, reaching a larger code means splitting the decoder across devices. The convolutional backbone is feedforward (acyclic), so it can be depth-split across multiple FPGAs at any layer boundary, the only inter-device traffic being the activation tensor flowing one way. Adding devices to fit a larger code (more layers for larger d) adds one fixed link-crossing delay per boundary to *reaction time*—a small, constant cost independent of code size—while leaving *throughput* unchanged, since the pipeline still accepts an input every cycle. Inter-device bandwidth is modest: a $d{=}19$, width-32 split transfers ~ 12.8 kB/round (~ 102 Gbps), a handful of standard 25 Gbps transceiver lanes.

This property is unavailable to iterative decoders. Belief propagation, matching, and union-find are inherently sequential—iteration $t{+}1$ consumes iteration t—so partitioning their graph forces a boundary-message round trip per iteration: for link latency δ and T iterations, decode time grows by $\sim T\delta$, degrading reaction time *and* throughput. The Relay-BP FPGA decoder Maurer et al. [2025] runs ~ 24 ns/iteration on one device; a two-device split with $\delta \sim 100$ ns round trip inflates this $\sim 4\times$, pushing a 12-iteration window from ~ 0.5 to ~ 2 μs—over budget before counting the larger code that motivated the split.

4 Discussion

We have shown that an accurate convolutional decoder can be made real-time on a single FPGA by co-design. Hardware decoders to date trade accuracy for speed: union-find on FPGA Liyanage et al. [2024], Collision Clustering on FPGA and ASIC Barber et al. [2025], and Relay-BP on FPGA Maurer et al. [2025] reach megahertz throughput, but as matching- or belief-propagation-based algorithms they forfeit the accuracy neural decoders attain in the below-threshold "waterfall" regime Gu et al. [2026]; neural decoders reach that accuracy but have so far run on throughput-oriented accelerators Senior et al. [2025]. Our design occupies the remaining corner—neural accuracy at deterministic single-chip latency, with a feedforward structure that scales across devices without sacrificing throughput.

However, several limitations remain. Our results cover memory experiments; decoding through logical operations—lattice surgery and transversal gates—is future work, requiring only compile-time changes to the convolution graph and routing. The outer RNN is a fully unrolled fp32 prototype that dominates the current footprint, which can be improved via hardened floating point DSPs.

References

Ben Barber, Kenton M. Barnes, Tomasz Bialas, Okan Buğdaycı, Earl T. Campbell, Neil I. Gillespie, Kauser Johar, Ram Rajan, Adam W. Richardson, Luka Skoric, Canberk Topal, Mark L. Turner, and Abbas B. Ziad. A real-time, scalable, fast and resource-efficient decoder for a quantum computer. *Nature Electronics*, 8(1):84–91, 1 2025. ISSN 2520-1131. doi: 10.1038/s41928-024-01319-5.

Sergey Bravyi, Andrew W. Cross, Jay M. Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J. Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005): 778–782, 3 2024. ISSN 0028-0836. doi: 10.1038/s41586-024-07107-7.

Tri Dao, Beidi Chen, Nimit S. Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 4690–4721, 2022. URL https://arxiv.org/abs/2204.00595.

Austin G. Fowler. Time-optimal quantum computation, 2013. URL https://arxiv.org/abs/1210.4626.

Andi Gu, J. Pablo Bonilla Ataides, Mikhail D. Lukin, and Susanne F. Yelin. Scalable neural decoders for practical fault-tolerant quantum computation, 2026. URL https://arxiv.org/abs/2604.08358.

Namitha Liyanage, Yue Wu, Siona Tagare, and Lin Zhong. Fpga-based distributed union-find decoder for surface codes. *IEEE Transactions on Quantum Engineering*, 5:1–18, 2024. ISSN 2689-1808. doi: 10.1109/tqe.2024.3467271.

Thilo Maurer, Markus Bühler, Michael Kröner, Frank Haverkamp, Tristan Müller, Drew Vandeth, and Blake R. Johnson. Real-time decoding of the gross code memory with FPGAs. 10 2025. URL http://arxiv.org/abs/2510.21600.

Andrew W. Senior, Thomas Edlich, Francisco J. H. Heras, Lei M. Zhang, Oscar Higgott, James S. Spencer, Taylor Applebaum, Sam Blackwell, Justin Ledford, Akvilė Žemgulytė, Augustin Žídek, Noah Shutty, Andrew Cowie, Yin Li, George Holland, Peter Brooks, Charlie Beattie, Michael Newman, Alex Davies, Cody Jones, Sergio Boixo, Hartmut Neven, Pushmeet Kohli, and Johannes Bausch. A scalable and real-time neural decoder for topological quantum codes. 12 2025. URL http://arxiv.org/abs/2512.07737.

Chang Sun, Zhiqiang Que, Vladimir Loncar, Wayne Luk, and Maria Spiropulu. da4ml: Distributed arithmetic for real-time neural networks on fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 19 (1), March 2026. ISSN 1936-7406. doi: 10.1145/3777387. URL https://doi.org/10.1145/3777387.

Barbara M. Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87 (2):307–346, 4 2015. ISSN 0034-6861. doi: 10.1103/RevModPhys.87.307.

Willers Yang, Jason Chadwick, Mariesa H. Teo, Joshua Viszlai, and Fred Chong. Spacetime-efficient and hardware-compatible complex quantum logic units in qldpc codes, 2026. URL https://arxiv.org/abs/2602.14273.