School of Computing and Information Systems
The University of Melbourne

Resource-Constrained FPGA Implementations of Surface-Code Neural Decoders

Verification, Experimental Evaluation and Scalable Integration

Mohit Garg
Student ID: 1681989
Master of Computer Science

Supervisors

1. **Prof. Lloyd C. L. Hollenberg**
   Melbourne Laureate Professor and Thomas Baker Chair, School of Physics
   Director, IBM Quantum Hub
   The University of Melbourne
2. **Prof. Rajkumar Buyya**
   Redmond Barry Distinguished Professor
   Director, Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory
   The University of Melbourne
3. **Dr Spiro Gicev**
   Quantum Error Correction Researcher, Diraq
   Research Associate, IBM Quantum Hub
   The University of Melbourne

November 2026

GitHub: https://github.com/Sp1cyJu1ce/fpga-qec-decoders

Abstract

Quantum error correction protects logical qubits through repeated rounds of parity checks. A classical decoder infers corrections from their outcomes. Published trapped-ion and neutral-atom experiments report millisecond-scale rounds. The superconducting processor Willow repeats its checks every 1.1 microseconds. On every platform, decoding must sustain the round rate and return corrections before dependent operations. Required code size depends on noise and the tolerated logical failure rate. This thesis implements two convolutional neural network (CNN) surface-code decoders on field-programmable gate arrays (FPGAs) and tests their scalability.

On an Arty A7-100T, every tested decision matched the integer software reference. At code distance 5 (81 physical qubits), banked memory and parallel arithmetic reduced spatial CNN inference per preloaded frame from 226.32 microseconds (shared-arithmetic baseline) to 38.73 microseconds. Distance 33 (4,225 physical qubits) took 1.154 milliseconds. For the history-based CNN, starting layers when their inputs became ready and increasing parallel hidden channels from 16 to 32 reduced a five-round distance-5 window from 5.569 milliseconds (fixed-slot baseline) to 2.354 milliseconds.

Each CNN proposes local corrections before global cleanup. Tests used simulated errors and offline ibm_miami records, accessed through the IBM Quantum Hub at the University of Melbourne. At distance 33 with perfect checks and 13% data noise, one integer CNN pass reduced a cluster decoder's failure rate from 27.0% to 13.3%. Minimum-weight perfect matching benefited less in spatial tests, and circuit-model matching often performed better without the CNN.

The improved Arty inference times are shorter than a reported 4.45-millisecond neutral-atom extraction round. For faster rounds, a larger FPGA completed distance-33 inference in 2,180 simulated cycles, or 21.8 microseconds at its routed clock. In simulation, the combined CNN and cluster decoder used fewer cycles than the cluster decoder alone at 13% noise, but more at 1% and 5%. Up to 64 simulated FPGA workers, each computing one region, reproduced undivided predictions through distance 65. A separate 64-FPGA model with 250 MHz clocks and 400 Gbit/s links projects 0.712 microseconds between slices of a loaded distance-33 history, before global cleanup. If sustained in a hardware stream, this rate would keep pace with Willow's 1.1-microsecond rounds.

Declaration

I certify that:

- this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- the thesis is 28,743 words in length (including the abstract and the prose and headings of Chapters 1–9; excluding captions, images, tables, equations, bibliographies and appendices).

Signature: _____ Date: _____

Planned publications

I am working towards translating the research contributions of this thesis into journal manuscripts. They will be further solidified through a research appointment at the IBM Quantum Hub at the University of Melbourne, under Prof. Lloyd C. L. Hollenberg. The following plan groups the five research questions addressed in the thesis into two papers.

1. Integer FPGA Implementations of Spatial and Space-Time Neural Surface-Code Decoders

Proposed journal: *Quantum Science and Technology*.

Research questions: RQ1, numerical correctness; RQ2, speed and resource limits.

This paper will present the integer representations and FPGA implementations of the two neural decoders. It will explain how memory banking, parallel arithmetic and layer scheduling reduce inference time while preserving the integer reference outputs. The evaluation will cover the measured latency and FPGA resource use on the Arty A7, numerical verification, and the limits imposed by memory capacity and bandwidth. The main material is in Chapters 4–6, with supporting checks in Appendices A, B and D.

2. Scaling FPGA Neural Decoders for Quantum Error Correction

Proposed journal: *IEEE Transactions on Quantum Engineering*.

Research questions: RQ3, complete-decoder quality; RQ4, connection to a quantum controller; RQ5, scaling through overlap and partitioning.

This paper will study when neural pre-decoding benefits the complete decoder and how its computation can be distributed across FPGAs. It will bring together decoding experiments on simulated errors and offline records from the IBM quantum processor. The hardware studies will cover connected cleanup tests, larger-FPGA implementations and distributed RTL simulations. The analysis will relate computation and communication costs to the timing requirements of different quantum platforms, distinguishing measured results from simulated behaviour and architecture projections. The main material is in Chapters 7 and 8, supported by Appendices B–D.

Notation and abbreviations

Symbol or short form	Full name and meaning
$[[n, k, d]]$	Code parameters. n data qubits encode k logical qubits with distance d; ancillas are counted separately.
d	Code distance. Fewest data qubits on which an error can act to cause an undetectable logical error.
R	Number of error-check rounds in one memory experiment.
p	Data-qubit depolarising-error probability in the ideal-check 2D model.
p'	Circuit-noise parameter in the 3D model, with operation-specific probabilities.
QEC	Quantum error correction. Protection of encoded information using error checks.
QPU	Quantum processing unit. The physical quantum processor.
CNN	Convolutional neural network. Shared local filters map input fields to predictions.
HDRG	Hard-decision renormalisation group. The cluster-based cleanup decoder used here.
MWPM	Minimum-weight perfect matching. Graph-based pairing of detection events.
PCM	Parity-check matrix. A binary map from error or correction bits to check changes.
DEM	Detector error model. Independent error mechanisms and their probabilities, including the correlated detection events produced by each mechanism.
FPGA	Field-programmable gate array. Reconfigurable digital logic, memory and routing.
ASIC	Application-specific integrated circuit. A chip manufactured for a particular task.
RTL	Register-transfer level. A description of clocked hardware and its data transfers.
CLB	Configurable logic block. A group of programmable logic and storage resources.
LUT	Lookup table. A programmable truth table used for logic or eligible small memories.
FF	Flip-flop. A one-bit register updated on a specified clock edge.
BRAM	Block random-access memory. Dedicated on-chip memory with fixed port modes.
URAM	UltraRAM. On-chip memory used for the feature maps in the distance-49 Versal design.
ROM	Read-only memory. Stores fixed values such as the network weights.
DSP	Digital signal processing block. Dedicated arithmetic hardware in the FPGA.
ReLU	Rectified linear unit. The activation function that replaces negative values with zero.
RQ	Requantisation. In network diagrams, ReLU followed by scaling, rounding and clipping. RQ1–RQ5 label the research questions.
II	Initiation interval. Time between accepted work units under a specified schedule.
WNS	Worst negative slack. The reported worst setup-time margin.
WHS	Worst hold slack. The reported worst hold-time margin.
FIFO	First-in, first-out. An ordered queue for temporarily stored records.
CRC	Cyclic redundancy check. A checksum used to detect corrupted records.
UART	Universal asynchronous receiver/transmitter. The laboratory serial interface.
ADC	Analogue-to-digital converter. Converts readout voltages into digital samples.
DAC	Digital-to-analogue converter. Converts digital samples into control waveforms.
RFSoC	Radio-frequency system-on-chip. Combines programmable logic and radio-frequency converters.
I/Q	In-phase and quadrature. Two components representing signal amplitude and phase.
XY4	A dynamical-decoupling sequence of four pulses about alternating X and Y axes, applied during an idle period.
TCO_H	Hidden-layer output-channel tile width in the 3D engines.
pp	Percentage points. The arithmetic difference between two percentages.
Kibit	Kibibit. $2^{10} = 1{,}024$ bits. One BRAM36 block holds 36 Kibit.
Mbit/s, Gbit/s	Megabits and gigabits per second. 10^6 and 10^9 bits per second.
Shot	One circuit execution, or one simulated trial where stated.
Round	One repeated extraction of the specified quantum checks.
Ancilla	A qubit used to measure one check. Also called a check qubit or measure qubit.
Syndrome	The ordered list of check outcomes. A residual syndrome is what remains after a proposed correction.
Detector	A set of measurement outcomes with a known parity in an error-free circuit. A deviation from that parity is a detection event.
Pauli frame	A classical record of inferred Pauli corrections used to interpret later operations and measurements.
Worker	A decoder circuit assigned a whole patch or one tile. Each modelled FPGA runs one worker in the distributed tests.
Tile	An arithmetic lane group in Chapter 6 or a region assigned to a worker in Chapter 8.
Halo	Neighbouring input values needed to calculate results near a tile's edges.

Contents

Abstract ii

Declaration iii

Planned publications iv

Notation and abbreviations v

1 Introduction 1
- 1.1 Quantum error correction and decoder timing . . . 1
- 1.2 Motivation for an FPGA-based convolutional decoder . . . 2
- 1.3 Research questions . . . 3
- 1.4 Summary of designs and findings . . . 4
- 1.5 Contributions and inherited work . . . 5
- 1.6 Thesis structure . . . 5

2 Background 6
- 2.1 Qubits, errors and checks . . . 6
- 2.2 The surface code . . . 7
- 2.3 Repeated measurements and detectors . . . 9
- 2.4 Decoders used in this thesis . . . 11
- 2.5 The two neural networks . . . 12
- 2.6 FPGA essentials . . . 15

3 Related work and the research gap 18
- 3.1 Decoding speed and resource costs . . . 18
- 3.2 Global decoders in previous work . . . 18
- 3.3 Neural decoders . . . 18
- 3.4 Neural-network inference on FPGAs . . . 20
- 3.5 Real-time decoding in hardware . . . 20
- 3.6 Parallel and distributed decoding . . . 21
- 3.7 The research gap . . . 21

4 Method: from a trained network to verified hardware 23
- 4.1 Overview . . . 23
- 4.2 Integer arithmetic for the 2D network . . . 23
- 4.3 Integer arithmetic for the 3D network . . . 24
- 4.4 Bit-width bounds . . . 25
- 4.5 Hardware verification . . . 26
- 4.6 Timing and resource measurement . . . 27
- 4.7 Measuring decoding quality . . . 28
- 4.8 Failure rates and confidence intervals . . . 30

5 The 2D decoder on the Arty A7 31
- 5.1 The workload . . . 31
- 5.2 From registers to addressed memory . . . 31
- 5.3 Banking and the arithmetic pipeline . . . 32
- 5.4 Scheduling and a cycle model . . . 35
- 5.5 Timing closure . . . 36
- 5.6 Results on the board . . . 37
- 5.7 Throughput and limits on further speed-up . . . 39

6 The 3D decoder on the Arty A7 40
- 6.1 Workload and memory requirements . . . 40
- 6.2 The input volume . . . 40
- 6.3 Mixed precision . . . 41
- 6.4 Four engines and channel banking . . . 42
- 6.5 Fixed-slot and readiness-based scheduling . . . 44
- 6.6 Results on the board . . . 45
- 6.7 Larger volumes and larger devices . . . 46

7 Decoding quality with the neural pre-decoder 48
- 7.1 2D simulations with perfect checks . . . 48

7.1.1 One CNN pass . . . 48
7.1.2 Multiple CNN passes . . . 50
7.1.3 Integer versus floating-point inference . . . 51
7.2 3D simulations with circuit noise . . . 52
7.2.1 Interpretation of the distance trends . . . 54
7.2.2 Multiple passes under circuit noise . . . 54
7.2.3 Sensitivity to the noise model . . . 55
7.3 Measurements from the IBM quantum processor . . . 55
7.3.1 Final-measurement experiments . . . 55
7.3.2 Repeated-check memory experiments . . . 57
7.3.3 Detector removal before cleanup . . . 59
7.4 Dependence on the trained network and cleanup decoder . . . 60
7.5 Implications for a hardware decoder . . . 61

8 System integration and scaling **62**
8.1 The path from qubit to correction . . . 62
8.2 Latency and throughput . . . 63
8.3 Quantum-computer interfaces and timing . . . 64
8.4 Input data rates . . . 66
8.5 Cleanup time with and without the CNN . . . 66
8.6 Partitioning the convolution across FPGAs . . . 68
8.7 Overlapping layers in a resident worker . . . 70
8.8 Distributing the computation . . . 71
8.9 Estimates for larger FPGAs . . . 73
8.10 Plateaus and steps in the scaling curves . . . 75
8.11 History buffering for a continuous stream . . . 76
8.12 Further integration work . . . 76

9 Conclusion **77**
9.1 Answers to the research questions . . . 77
9.2 Limitations . . . 79
9.3 Future work . . . 79
9.4 Closing remarks . . . 80

References **81**

A Numerical details **84**
A.1 Derivation of the bounds . . . 84
A.2 2D network . . . 84
A.3 3D network . . . 85

B Implementation details **86**
B.1 The board interface . . . 86
B.2 Layer scheduling checks . . . 87
B.3 Arty timing and capacity limits . . . 87
B.4 Additional classifier check . . . 87
B.5 Components implemented on larger FPGAs . . . 88
B.6 Partitioning and the resident worker . . . 88
B.7 The distributed RTL experiment . . . 89
B.8 Global cleanup on an FPGA . . . 90

C Supporting decoding-quality results **93**
C.1 Pruning with complete decoding . . . 93
C.2 Local prediction sensitivity with defects . . . 94
C.3 Final-measurement records from the IBM quantum processor . . . 94
C.4 Distance comparison on device measurements . . . 95
C.5 The repeated-check records . . . 97
C.6 Device layouts . . . 99

D Supplementary validation **100**
D.1 Integer and RTL checks . . . 100
D.2 Fresh-sample decoding check . . . 100

Acknowledgements **101**

List of Tables

2.1 Effect of one incorrect readout on the detection events . . . 11
3.1 Timing boundaries in published and measured decoder experiments . . . 21
4.1 Examples of faults found during verification . . . 27
5.1 2D measurements on the Arty A7 . . . 37
6.1 3D measurements on the Arty A7 . . . 45
7.1 Distance-33 failure counts in the 3D simulation . . . 53
8.1 CPU cleanup time before and after the CNN . . . 67
8.2 Resident CNN with HDRG and HDRG alone at four error probabilities . . . 71
8.3 Distance-33 completion in simulation . . . 73
A.1 2D arithmetic bounds . . . 84
A.2 3D arithmetic bounds . . . 85
B.1 Resources on larger FPGAs . . . 88
B.2 Internal timing on larger FPGAs . . . 88
B.3 Distance-33 communication traffic . . . 89
B.4 Standalone cleanup circuits . . . 91
B.5 Clock cycles for the connected CNN and HDRG . . . 92
C.1 Complete decoding after weight pruning . . . 94

List of Figures

1.1 A surface code, a bit-flip error and its correction . . . 2
1.2 Three decoders on the same four fired checks . . . 3
2.1 Locating a bit-flip error with two parity checks . . . 7
2.2 The unrotated distance-5 surface code . . . 8
2.3 Different logical outcomes from the same syndrome . . . 10
2.4 Spatial syndromes versus detector histories . . . 11
2.5 Measuring checks and combining local and global corrections . . . 13
2.6 The two convolutional decoder networks . . . 14
2.7 FPGA building blocks on the Arty A7 . . . 16
2.8 Latency, initiation interval and waiting inputs . . . 17
4.1 Integer scales and the layer-two datapath in the 2D network . . . 24
4.2 Decoder combinations and their evaluation . . . 28
4.3 Finite detector windows and the 3D decoders . . . 29
5.1 Storage and scheduling in the three 2D architectures . . . 32
5.2 Reading consecutive pixels from four memory banks . . . 33
5.3 Aligned data and control in the layer-two engine . . . 34
5.4 Cycle breakdown for the overlapping 2D schedule . . . 36
5.5 Measured 2D inference time on the Arty A7 . . . 38
5.6 2D core cycles by code distance and lane count . . . 38
5.7 Resource use of the banked 2D design by code distance . . . 39
6.1 Unit cell and channels of the 3D network . . . 41
6.2 Effect of weight precision on 3D predictions . . . 42
6.3 The four concurrent 3D engines . . . 43
6.4 Block-memory mapping in the 3D design . . . 44
6.5 Measured 3D core time and simulated cycle counts . . . 46
7.1 Effect of one CNN pass on 2D decoding failures . . . 49
7.2 Effects of CNN pass count and precision on failure rate . . . 51
7.3 Effect of one CNN pass on 3D memory failures . . . 53
7.4 Final-measurement experiments on the IBM quantum processor . . . 56
7.5 Error per site during idle time on ibm_miami . . . 57
7.6 Decoder failure rates on final-measurement records from the IBM quantum processor . . . 58
7.7 Effect of one CNN pass on repeated-check records from ibm_miami . . . 59
7.8 Detector removal on repeated-check records from ibm_miami . . . 60
8.1 Superconducting readout and the path to a correction . . . 62
8.2 Check records and the context for one 3D prediction . . . 64
8.3 Circuit durations compiled for ibm_miami . . . 65
8.4 From qubit readout to a decoder input . . . 65
8.5 Owned regions, halos and the cost of overlap . . . 69
8.6 Overlapping layers in the resident worker . . . 70
8.7 2D worker partitions and simulated frame time . . . 72
8.8 Model estimates for one 2D frame . . . 74
8.9 Model estimates for 3D volumes and slices . . . 75
A.1 Required and allocated bit widths in the 2D datapath . . . 85
B.1 The board test interface . . . 86
B.2 Issue schedule of the 2D layers . . . 87
B.3 FPGA circuits for global cleanup after the CNN . . . 90
C.1 Magnitude pruning with complete decoding . . . 93
C.2 Effect of pruning on local CNN predictions . . . 95
C.3 Bell outcome frequencies and correlations . . . 96
C.4 Decoder failures at code distances 5 to 8 . . . 96
C.5 Decoding repeated-check records from ibm_miami . . . 97
C.6 Detector activity on ibm_miami . . . 98
C.7 Detector firing rates across the patch . . . 98
C.8 Recorded qubit placements for repeated checks . . . 99

1. Introduction

In a classical computer, transistors act as switches to represent bits as 0 or 1. A quantum computer uses qubits, which can be prepared in a superposition of these two states. For some problems, a quantum algorithm can reach an answer in exponentially fewer steps. The advantage depends on how the input is supplied and what the algorithm must return [1]. Researchers are exploring applications such as modelling molecules in pharmaceutical research, pricing options in finance and classifying data in machine learning [2, 3, 4]. Running these calculations reliably requires protecting quantum information from errors. This thesis studies the classical hardware that interprets the quantum processor's pattern of check outcomes, called a *syndrome*, to determine the required correction.

1.1 Quantum error correction and decoder timing

A quantum computer stores information in physical systems that are sensitive to noise. Gates, idle periods and measurements can introduce errors. Quantum error correction (QEC) protects information by encoding a *logical qubit* across many physical *data qubits*. Additional qubits, called *ancillas*, measure checks on groups of neighbouring data qubits. Measuring these checks gives information about errors without revealing the encoded value [5, 6]. A check is said to *fire* when its outcome differs from its expected value.

The notation $[[n, k, d]]$ describes a code with n data qubits, k logical qubits and distance d. The distance is the smallest number of data qubits on which an error can act to change the logical information while leaving all check outcomes unchanged. With perfect checks, any error on fewer than d qubits either changes the syndrome or acts harmlessly on the encoded state. When the error locations are unknown, successful correction is guaranteed for up to $\lfloor (d-1)/2 \rfloor$ affected qubits [6].

Figure 1.1(a) shows a $[[13, 1, 3]]$ surface code [5, 7]. Thirteen data qubits store one logical qubit. The twelve ancillas used to measure its checks are counted separately. Flipping all three ringed qubits changes the logical state without firing a check, yielding a *trivial syndrome* in which every check has its expected outcome. A single flip at the centre gives a different result. The two adjacent checks fire (b). Under the assumption of a single error, the decoder can identify the affected qubit. The controller can then keep a record of the correction and use it to interpret a later measurement (c). For more general error patterns the syndrome can have several explanations.

Timing remains a constraint even when the controller can wait before using a correction. Google's Willow processor produces a new round of measurements every 1.1 μs [8]. A decoder can take longer than one round to finish a correction and still match the arrival rate if it processes several rounds concurrently. If its average processing rate falls below the arrival rate, rounds accumulate and later operations must wait [6].

Increasing distance improves protection when the physical error rate is below the threshold for the code, noise model and decoder. The patch also grows in two dimensions, so the number of checks per round grows approximately as d^2. A local decoder can process different regions at the same time. Whether it can finish a larger patch in the same time depends on how much hardware is available and how quickly the measurements can reach it.

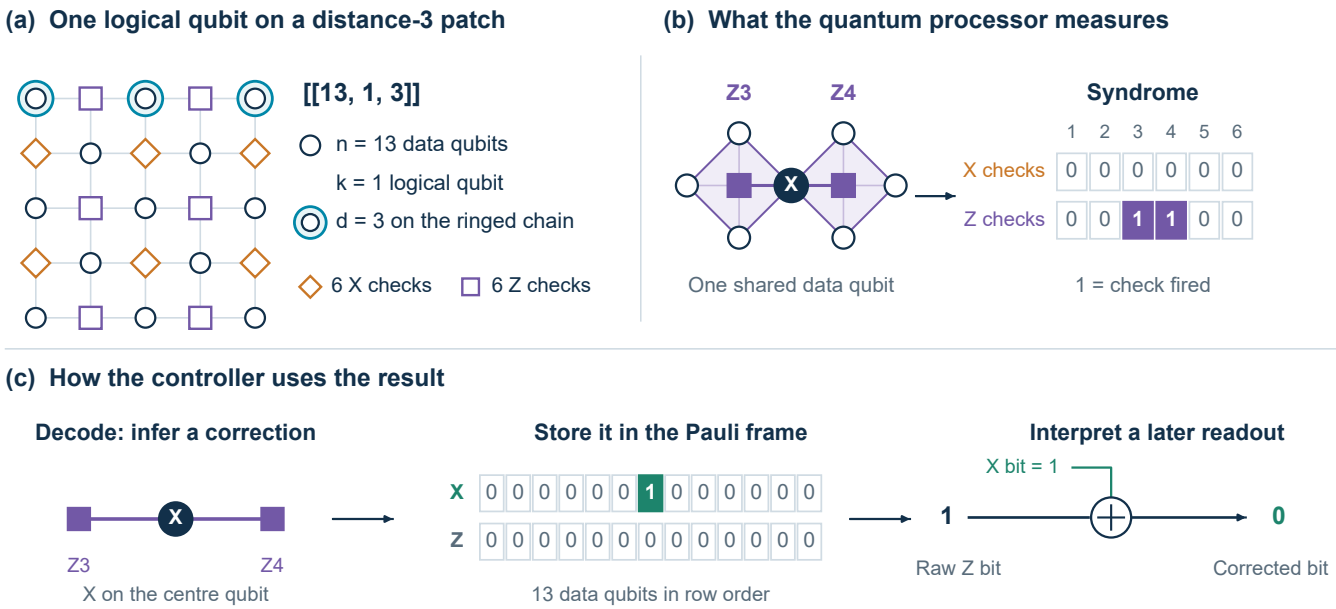


Figure 1.1: A surface code, a bit-flip error and its correction

(a) The $[[13, 1, 3]]$ patch stores one logical qubit in 13 data qubits. Twelve ancillas measure the checks, which involve four data qubits in the interior and three at the boundaries. Flipping the three ringed data qubits changes the logical state without changing any check. (b) A single X error at the centre changes two Z checks. Each shaded diamond shows the four data qubits measured by one of these checks. The bits for each check type are listed in row order. (c) Assuming perfect measurements and at most one error, the decoder infers from checks Z3 and Z4 that their shared data qubit was flipped. The controller records the X correction in the Pauli frame. At a later Z readout of that qubit, the stored bit reverses the interpretation of the measured bit, here from 1 to 0.

1.2 Motivation for an FPGA-based convolutional decoder

Minimum-weight perfect matching (MWPM) is a standard reference decoder for the surface code. It connects fired checks to one another or to a suitable boundary, choosing paths of minimum total weight (Figure 1.2(a)). Its workload depends on the pattern of checks that fire. At 0.1% circuit noise, Sparse Blossom took under 1 µs per round at distance 17 and 3.5 µs at distance 29 on a single CPU core [9]. These are average times per round in a batch. In the distance-33 experiment in Table 8.1, at 0.5% circuit noise, the same family of algorithms took about 1.9 ms per 33-round input in each measurement basis. A simpler global decoder, hard-decision renormalisation group (HDRG), groups nearby fired checks and expands its search when a group cannot be resolved (Figure 1.2(b)).

The convolutional neural networks (CNNs) developed by Gicev et al. apply the same learned weights to a fixed neighbourhood around each output [10, 11]. The arithmetic stays the same whether one check fires or many, and distant regions can be processed in parallel. The CNN acts as a *pre-decoder*. It proposes local corrections and sends the remaining pattern to a global decoder (Figure 1.2(c)). The comparison tests whether the combination makes fewer logical errors and whether the time saved in the global search exceeds the time spent on the CNN.

A field-programmable gate array (FPGA) is a chip whose logic, memory and wiring can be configured for a particular computation. It suits the CNN because several copies of the same arithmetic circuit can process neighbouring regions in parallel. During inference, the network applies its fixed, trained weights to a new input. A larger patch reuses the same weights at more positions. The CNN cores developed here take a fixed number of clock cycles for a given input size. FPGAs are also part of quantum control systems, close to where measurements become available [12, 13].

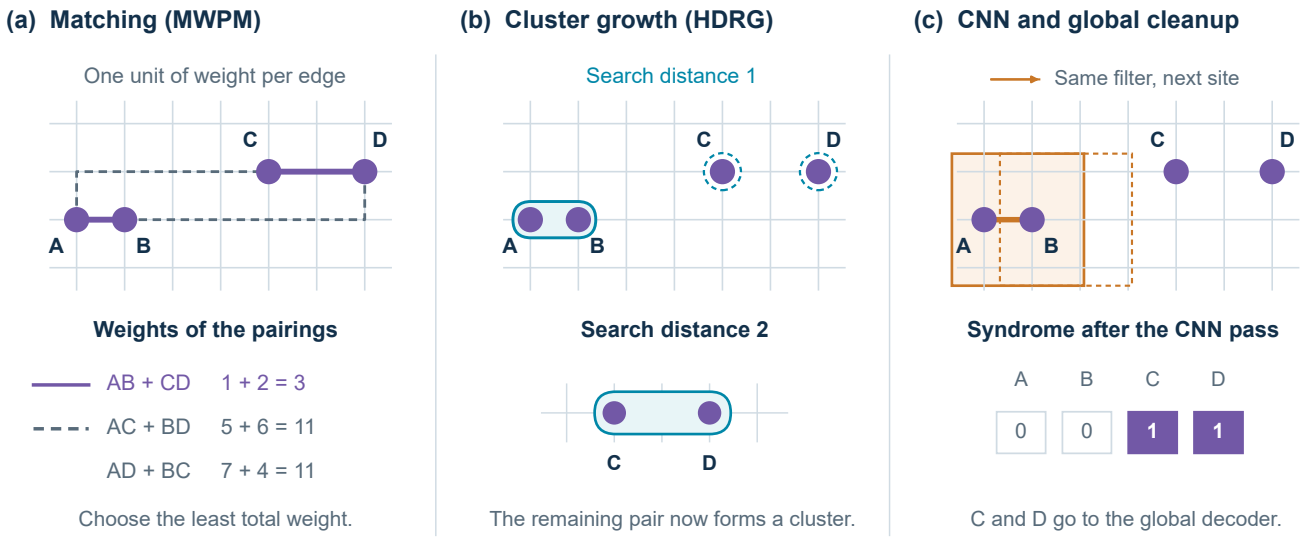


Figure 1.2: Three decoders on the same four fired checks

(a) The four filled dots lie in the interior of a patch with equal edge weights. Minimum-weight perfect matching (MWPM) selects the two solid paths with total weight 3. The dashed paths show one of the two alternatives with weight 11. (b) HDRG clusters checks A and B at search distance 1. Checks C and D remain separate until distance 2. (c) The CNN applies the same filter at each position. The shaded window and its dashed copy illustrate the shift by one site. One possible local correction clears A and B, leaving C and D for the global decoder. The window is schematic.

This thesis uses two models. The *2D* model uses one spatial snapshot of the checks and assumes the checks themselves are measured perfectly. The *3D* model also uses the history of repeated, noisy measurements, so its third axis is time. The experiments use the original weights to measure inference time and test whether the predicted corrections improve decoding.

The Digilent Arty A7-100T provides a small, inexpensive platform for these tests [14]. Its limited arithmetic and memory help identify the main sources of delay. Two circuits can run the same network and give exactly the same answers, yet one may take longer because several calculations must share a multiplier or wait for a value from memory. The board experiments vary one design choice at a time and measure its effect in clock cycles. Appendix B.1 describes the board interface and how its timings are measured. The larger-device studies then examine how those choices change when more resources are available.

1.3 Research questions

To what extent can FPGA implementations of surface-code neural decoders scale to larger code distances while maintaining numerical correctness and error-correction performance, and meeting the latency and throughput requirements of quantum error correction?

The spatial FPGA experiments use an unrotated surface code patch with $d^2 + (d-1)^2$ data qubits and one ancilla for each check. At distance 5 this is 41 data qubits and 40 ancillas. At distance 33 it is 2,113 data qubits and 2,112 ancillas. The Arty board processes check patterns for patches through distance 33. The distributed simulation divides the calculation among FPGA workers, each a circuit assigned part of the patch, and reaches distance 65. A simulation with one worker reaches distance 97. These correspond to patches of 16,641 and 37,249 physical qubits respectively.

The required size of a surface code patch depends on its noise and the logical error rate that the computation can tolerate. As one example, two estimates for factoring a 2,048-bit RSA integer use distances 27 and 25 for the working patches. Both assume a 0.1% physical gate error rate, a 1 µs round and a 10 µs response from the classical control system [15, 16]. The five research questions below ask which parts of such a timing budget the FPGA can address.

RQ1: numerical correctness. How should a trained floating-point network be represented using integers, and what test shows that the hardware reproduces this calculation exactly? (Chapter 4)

RQ2: speed and resource limits. On a small FPGA, which resources limit speed and code distance, and how far can memory organisation, parallel arithmetic and scheduling extend those limits? (Chapters 5 and 6)

RQ3: complete-decoder quality. Which cleanup decoders make fewer logical errors when paired with the CNN? (Chapter 7)

RQ4: connection to a quantum controller. What timing and data-rate requirements connect the accelerator to a quantum controller? (Chapter 8)

RQ5: scaling through overlap and partitioning. How far can overlapping layers and distributing work across devices reduce decoding time as the code grows? (Chapter 8)

For numerical correctness, the reference is the integer software model of each network. For speed, each board design is timed against the first implementation on the same FPGA. For decoding quality, the CNN-assisted decoder and the global decoder alone receive the same trials. Table 3.1 places the hardware timings alongside published results and identifies which part of the decoding process each study timed. Section 9.1 returns to the five questions in order.

1.4 Summary of designs and findings

- **Faster 2D inference on a small FPGA.** The first board-verified design took 226.32 µs to process one $d = 5$ patch. Banked memory, parallel arithmetic and independent layer scheduling lowered this to 49.38 µs. More parallel arithmetic at 90 MHz reduced it further to 38.73 µs, 5.84 times faster than the first design. The same board runs distance 33 in 1.154 ms (Table 5.1).
- **A 3D implementation enabled by mixed precision.** Storing all 971,568 weights at eight bits would need 156% of the board's block memory. Four-bit weights in the two large middle layers reduce the space needed for the weights to 85.7% of the board's block memory. Readiness-based scheduling and more parallel arithmetic reduce a distance-5, five-round inference from 5.569 ms to 2.354 ms. Across the seven 2D and two 3D configurations in the repeated-transaction campaign, all 27,086,000 decision bits returned in 9,000 transactions matched the integer reference (Sections 5.6 and 6.6).
- **The CNN's benefit depends on the cleanup decoder.** At distance 33 and a physical error probability of 13% in the ideal-check study, one floating-point CNN pass reduces HDRG's failure rate from 27.0% to 12.9%. MWPM shows a smaller reduction, from 5.49% to 5.13%. Correlated matching already fails on only 1.25% of trials, and adding the CNN increases its failure rate. In the space-time simulations and on records from the IBM quantum processor, preprocessing often helps simple cleanup methods but increases failures for matching methods that use more detailed noise information (Chapter 7).

- **Communication limits distributed scaling.** Each FPGA worker processes one tile of the patch and exchanges values near its edges with neighbouring workers. Splitting the 2D computation across up to 64 simulated workers reproduces the undivided result exactly, up to distance 65. Adding workers helps only while the reduction in computation outweighs communication and scheduling costs. Under the tested fixed total bandwidth of 32 Gbit/s, four workers were fastest at every sampled distance up to 49 (Section 8.8).

1.5 Contributions and inherited work

The neural network architectures, their training and the idea of combining a local network with a global decoder belong to their original authors [10, 11, 17]. The contributions of this thesis are:

1. integer formats with proven bit widths for both networks, and a verification flow that checks every intermediate value of the hardware against them;
2. resource-constrained FPGA architectures for both networks (banked memories, lane parallelism and readiness-based scheduling), physically validated on the Arty A7-100T;
3. paired comparisons of complete decoders on the same simulated errors and on measurement records from the IBM quantum processor `ibm_miami`, accessed through the IBM Quantum Hub at the University of Melbourne, including MWPM and correlated matching on spatial syndromes, and matching graphs built from parity checks or circuit noise for measurement histories;
4. a connected spatial CNN and HDRG circuit tested through distance 33, and an analysis of the timing and data rates a quantum controller would need; and
5. resident and distributed RTL designs that test overlapping layers and multi-device partitioning, with selected components implemented on larger FPGAs.

Code and coefficient exports are in the GitHub repository.

1.6 Thesis structure

Chapters 2–4 introduce the background, earlier work and experimental methods. Chapters 5 and 6 describe the two FPGA implementations. Chapter 7 evaluates whether their predictions improve decoding. Chapter 8 examines the cost of connecting a decoder to a quantum computer and dividing its work across devices. Chapter 9 summarises the results and answers the research questions. The appendices give the numerical bounds (Appendix A), further implementation results (Appendix B), supporting decoding results (Appendix C) and independent checks (Appendix D).

The Arty board supplied the measurements for both CNN cores. For larger designs, register-transfer-level (RTL) simulation checks the behaviour of the circuit in every clock cycle, and Vivado timing analysis checks whether a routed circuit can run at the stated clock frequency. Each timing estimate states the assumptions behind it. Chapter 8 compares the rate of incoming checks with the rate the hardware can sustain, and gives the time needed to finish one input. For decoding quality, each CNN-assisted decoder is compared with the same global decoder on identical simulated errors or device measurements. Chapter 7 describes how the circuits on the IBM quantum processor were run and how their records were decoded offline.

2. Background

This chapter introduces quantum error correction, the surface code and its decoders. It then describes the two neural networks implemented in this thesis and the FPGA concepts that determine how quickly they can run.

2.1 Qubits, errors and checks

A qubit has two computational basis states, written $|0\rangle$ and $|1\rangle$. A pure qubit state can be written as a superposition $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, with complex amplitudes $\alpha, \beta \in \mathbb{C}$ satisfying $|\alpha|^2 + |\beta|^2 = 1$. Measuring it in the computational basis returns 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$, leaving it in the corresponding state. One measurement gives a bit and cannot reveal both amplitudes [18].

Two gates help explain the circuits used later. The Hadamard gate (H) maps $|0\rangle$ to $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|1\rangle$ to $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. On computational-basis states, a controlled-NOT (CNOT) flips its target bit when its control bit is 1. Starting with $|00\rangle$, applying Hadamard to the first qubit and then a CNOT from the first to the second prepares the Bell state $(|00\rangle + |11\rangle)/\sqrt{2}$. The qubits of this state are entangled, which means their joint state cannot be described as two independent pure states. Computational-basis readout gives 00 or 11 with equal probability. Those outcomes alone cannot distinguish the Bell state from a classical mixture that produces 00 and 11 equally often. Measurements in another basis can distinguish these cases. Section 7.3.1 explains how the Bell experiments use an inverse preparation circuit before readout to extract bit-like and phase-like error information.

Errors on a qubit can be described, for the purposes of error correction, by the Pauli operators. An X error exchanges $|0\rangle$ and $|1\rangle$ (a *bit flip*). A Z error changes the sign of the $|1\rangle$ component (a *phase flip*). This leaves computational-basis probabilities unchanged but can affect a measurement in another basis. For example, a Z error exchanges the two X-basis states $|+\rangle$ and $|-\rangle$. A Y error combines a bit and phase flip, apart from an overall phase that does not affect measurement probabilities. Real noise is more varied, but correcting every Pauli error supported on a given set of qubits is sufficient to correct an arbitrary error on that set [6].

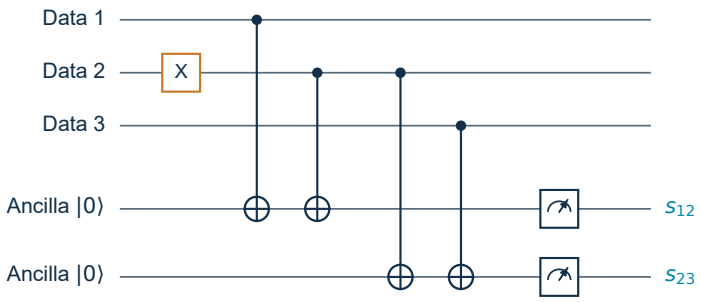
Classical error correction can copy a bit and take a majority vote. The no-cloning theorem rules out a procedure that can perfectly copy an arbitrary unknown quantum state [19]. Reading the encoded value directly would also generally disturb its superposition. Instead, quantum codes measure *parities* of several qubits. Consider two qubits in the state $\alpha|00\rangle + \beta|11\rangle$. The operator Z_1Z_2 measures whether their computational-basis values agree and returns $+1$ for either $|00\rangle$ or $|11\rangle$. The outcome is the same for both terms of the superposition, so the measurement reveals nothing about α or β. If an X error flips one qubit, the outcome becomes -1. The check detects this error without reading the protected information.

Each such parity measurement is a *check*. Its outcome is recorded as a bit, 0 for $+1$ and 1 for -1. A check with outcome 1 is said to have *fired*. The list of check outcomes, in a fixed order, is the *syndrome*.

In Figure 2.1, the three-qubit repetition code encodes $\alpha|0\rangle + \beta|1\rangle$ across data qubits 1, 2 and 3 as $\alpha|000\rangle + \beta|111\rangle$. Two checks, Z_1Z_2 and Z_2Z_3, compare neighbouring pairs. If the middle qubit suffers an X error, both checks fire. An end-qubit flip changes only one check. Assuming at most one bit flip, the syndrome identifies which qubit to flip back.

With two or more flips the same outcomes can have another explanation (Section 2.4). In this example, phase flips leave both checks unchanged. Other codes, such as the surface code, also use checks composed of Pauli X operators to detect phase flips.

(a) Two parity measurements



(b) Single bit-flip lookup

s_{12}	s_{23}	Correction
0	0	None
1	0	Flip data 1
1	1	Flip data 2
0	1	Flip data 3

0 = even parity 1 = odd parity

Figure 2.1: Locating a bit-flip error with two parity checks

(a) The box marked X represents a bit-flip error on the middle data qubit. Two ancillas compare neighbouring pairs of data qubits. On each CNOT, the filled dot is the control and the circled plus is the target. (b) With perfect checks and at most one bit flip, the two readouts identify the qubit to correct. Both checks fire for a flip on the middle qubit, as in the highlighted row.

A *physical error* acts on one or more physical qubits. Its *weight* is the number of qubits on which it acts nontrivially. The *code distance* d is the smallest weight of an undetected Pauli error that acts on the encoded information as a *logical error*. In a classical binary code the distance is the fewest bit flips needed to turn one valid codeword into another. A quantum code must protect the logical states $|0\rangle$ and $|1\rangle$ against a bit flip and must also protect their relative phase when they are in superposition. A logical bit or phase flip leaves every check outcome at zero, so the syndrome alone does not reveal the logical error. Decoding succeeds if the combined effect of the physical error and its correction is harmless to the encoded state.

These parity-check operators are called *stabilisers*. Encoded states are $+1$ eigenstates of each stabiliser: measuring that stabiliser returns $+1$ without changing the state. Applying a stabiliser, or a product of stabilisers, leaves such a state unchanged. Consequently, two different physical corrections can both succeed if they differ by a product of stabilisers. After either correction, the encoded information is the same.

2.2 The surface code

The surface code arranges data qubits and two kinds of checks on a grid [5]. Each Z-type check measures the joint parity of up to four neighbouring data qubits and fires when an odd number of them have X or Y errors. Each X-type check does the same for Z or Y errors. The rule can be expressed using *anticommutation*. A Pauli error E reverses the outcome of a check S when $SE = -ES$, meaning that exchanging the order of the two operators changes the sign of their product. If $SE = ES$, the outcome is unchanged. A single-qubit Y error anticommutes with both X and Z, so it reverses both kinds of check that touch that qubit. Each check is measured by gates between an ancilla and nearby data qubits. These local gates suit hardware whose qubits interact only with their neighbours.

The 2D network uses the *unrotated* distance-5 patch in Figure 2.2. Data and check positions are interleaved on a $(2d-1) \times (2d-1)$ grid, here 9×9. Data qubits occupy sites whose row and column indices have the same parity.

X checks occupy sites on odd rows and even columns, and Z checks occupy sites on even rows and odd columns. The patch has $d^2 + (d-1)^2 = 41$ data qubits and $2d(d-1) = 40$ checks. In a circuit with one ancilla per check, 40 ancillas measure those parities, so the patch uses 81 physical qubits. An interior check touches four data qubits and a boundary check touches three. The shortest error chains that cross the patch without firing any check have length five, which is why $d = 5$.

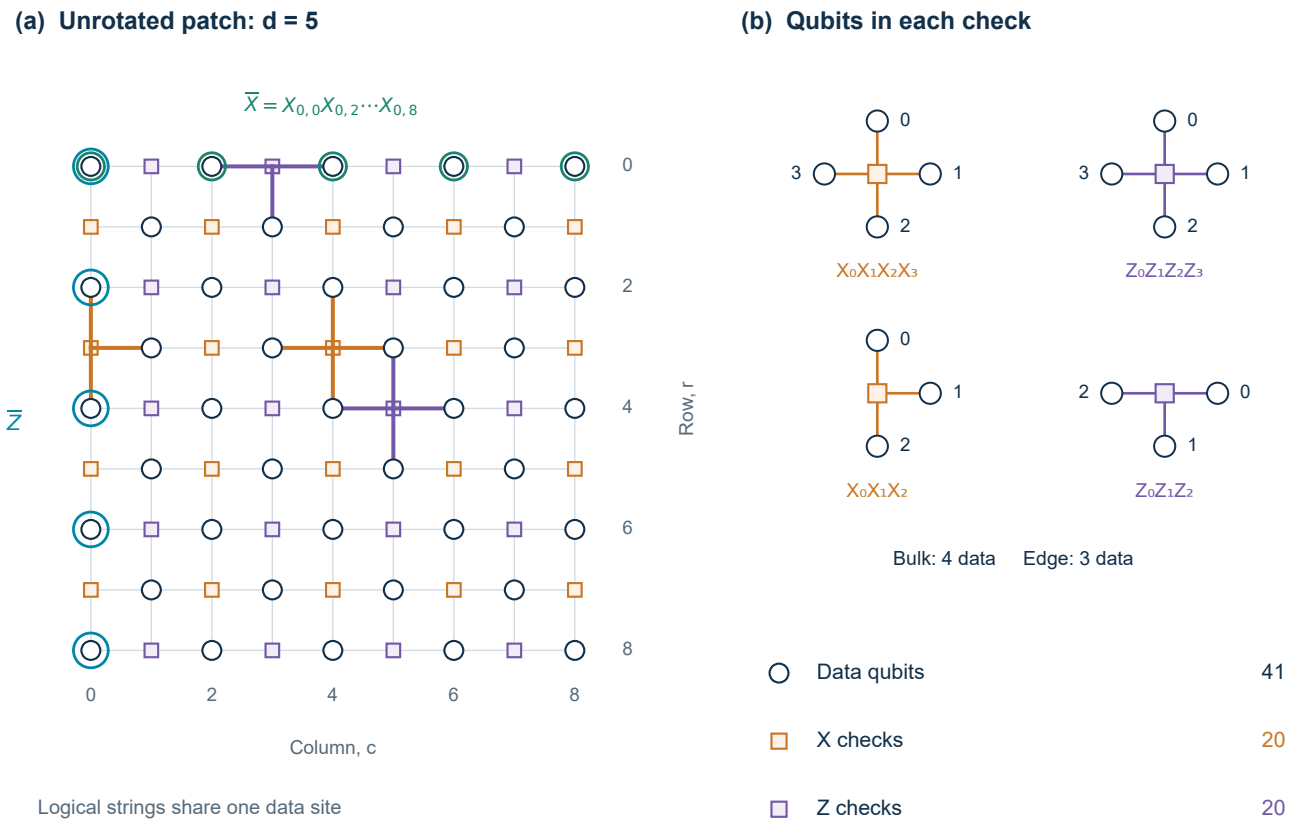


Figure 2.2: The unrotated distance-5 surface code

(a) The patch on its 9×9 interleaved grid: 41 data qubits (circles), 20 X checks and 20 Z checks (squares), with lines joining each check to the data qubits it measures. Weight-five logical X and Z strings cross the patch. (b) Interior checks act on four data qubits and boundary checks on three. The 3D network uses the rotated layout instead.

To see why two checks can indicate a single error, consider an even-numbered row of data qubits in Figure 2.2. An X on an interior data qubit reverses the parity of both neighbouring Z checks along that row. If the next data qubit in the row also flips, their shared check sees two flips and returns to its original value. Only the two outer checks still fire. A longer unbroken string produces the same pattern at its ends. At some edges of the patch there is no further check of the relevant type, so a string can also leave only one visible end. The decoder observes these ends and must infer a plausible path between them or to the boundary.

The 3D network uses the *rotated* surface code instead [20], which needs only d^2 data qubits and $d^2 - 1$ checks for the same distance: 25 and 24 at $d = 5$. A distance-5 result therefore refers to 41 data qubits in the 2D experiments and 25 in the 3D experiments.

A larger patch contains more locations at which a physical error can occur, but a longer chain of errors is needed to change the logical state. The *threshold* is the physical error rate below which increasing the distance suppresses logical errors for a given code, noise model and decoder. Under a simple independent-error model, the logical failure

rate then decreases approximately in proportion to $(p/p_{\text{th}})^{\lceil d/2 \rceil}$, where p is the physical error probability and p_{th} is the threshold [5]. Above threshold, increasing distance can instead make failure more likely. Google's Willow experiment demonstrated suppression in hardware, with the logical error per round falling by a factor of more than two each time the distance grew from 3 to 5 to 7 [8]. Changing the decoder can change this balance even when the physical noise stays the same. A few finite-distance results can show this trend, but estimating a threshold requires comparisons across distances and noise levels. Reliable decoding and sufficient processing speed are separate requirements, both of which this thesis measures.

Two different corrections can both clear the checks, yet only one may preserve the logical state (Figure 2.3). To describe their combined effect, each Pauli operator can be represented by two bits: $I = [0, 0]$, $X = [1, 0]$, $Y = [1, 1]$ and $Z = [0, 1]$. The bits record the presence of an X component and a Z component. Bitwise exclusive OR, written $\oplus$, combines two such patterns, ignoring an overall phase that does not affect the encoded information. Applying the same Pauli error twice cancels it, just as the XOR of a bit with itself is zero.

The actual error E and the two candidate corrections C_1 and C_2 in the figure each correspond to the same syndrome. Applying either correction after E therefore clears the checks. However, the combined pattern $E \oplus C_1$ is a product of stabilisers and leaves the encoded state unchanged, whereas $E \oplus C_2$ stretches across the patch and causes a logical error. Successful decoding must both clear the syndrome and preserve the logical information.

2.3 Repeated measurements and detectors

In the repeated-check circuits studied here, each check uses an additional *ancilla* qubit. Gates between the ancilla and its neighbouring data qubits transfer their joint parity onto the ancilla, which can then be read while the data qubits continue to store the encoded state [5]. A readout error can make a check appear to fire, so the circuit repeats these measurements in *rounds*. The decoder then uses *changes* between rounds rather than raw check values. One prepared experiment, called a *shot*, can contain many rounds before the final readout. Repeating shots estimates how often the experiment fails. Section 7.3.2 applies this distinction to the repeated-check experiments on the IBM quantum processor.

A *detector* specifies a set of measurement outcomes whose combined parity is known in an error-free circuit. A departure from that expected parity is a *detection event*. For a repeated check in the interior of a memory circuit, this usually tests whether its parity changed between rounds. A persistent data error can change the parity when it first appears. A readout error can also cause detection events even though the data qubits are unchanged. Figure 2.4 shows how repeated measurements give the decoder a time axis as well as a spatial pattern.

A short measurement history illustrates what the time axis adds. Suppose a reset-based check reads 0, 0, 1, 1, 1 over five rounds. Its parity changed once, at round three. The repeated 1s are consistent with an error that persists on the data. Those repeated 1s therefore represent one detection event. In a circuit with many checks, the spatial positions of the other changes help distinguish this explanation from an error in an ancilla readout.

The rule for forming detectors depends on the measurement circuit. Most commonly, each ancilla is reset after measurement, so an interior detector is $D_t = m_t \oplus m_{t-1}$ for raw outcomes m_t. The repeated-check memory circuits on the

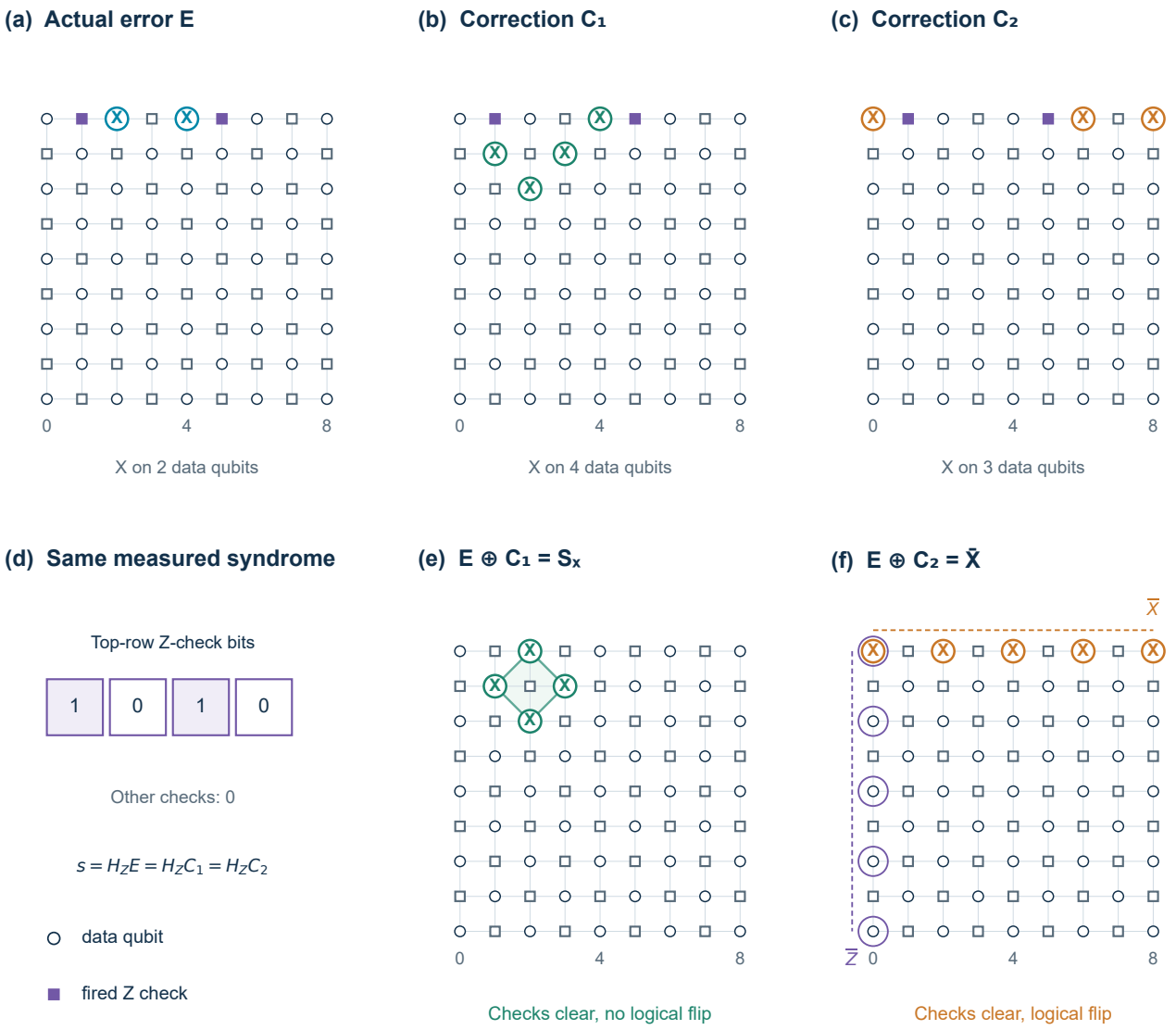


Figure 2.3: Different logical outcomes from the same syndrome

(a–c) An X error E and two possible corrections C_1 and C_2 act on two, four and three data qubits, respectively. (d) All three fire the same two top-row Z checks. Both corrections therefore clear the syndrome. (e) The four qubits left in $E \oplus C_1$ form one X stabiliser and leave the logical state unchanged. (f) $E \oplus C_2$ crosses the patch and intersects the marked logical Z string once, acting as a logical X error.

IBM quantum processor `ibm_miami` measure the ancillas after every round and reuse them without a reset between rounds. Without a reset, the ancilla starts each round in the state left by the previous measurement. The inferred check value is therefore $s_t = m_t \oplus m_{t-1}$. Comparing two successive check values gives the interior detector

$$D_t = s_t \oplus s_{t-1} = m_t \oplus m_{t-2}.$$

Using the reset rule on no-reset data would produce incorrect decoder inputs from correct measurements. The first and last rounds need their own rules, because they compare against the prepared state and against the final readout of the data qubits.

Table 2.1 follows one readout error. The true parity stays at zero, but the bit recorded in round three is wrong. Both circuits flag round three. The reset circuit flags round four as well because it compares that round with round three. Without reset, round five is the next reading compared with round three.

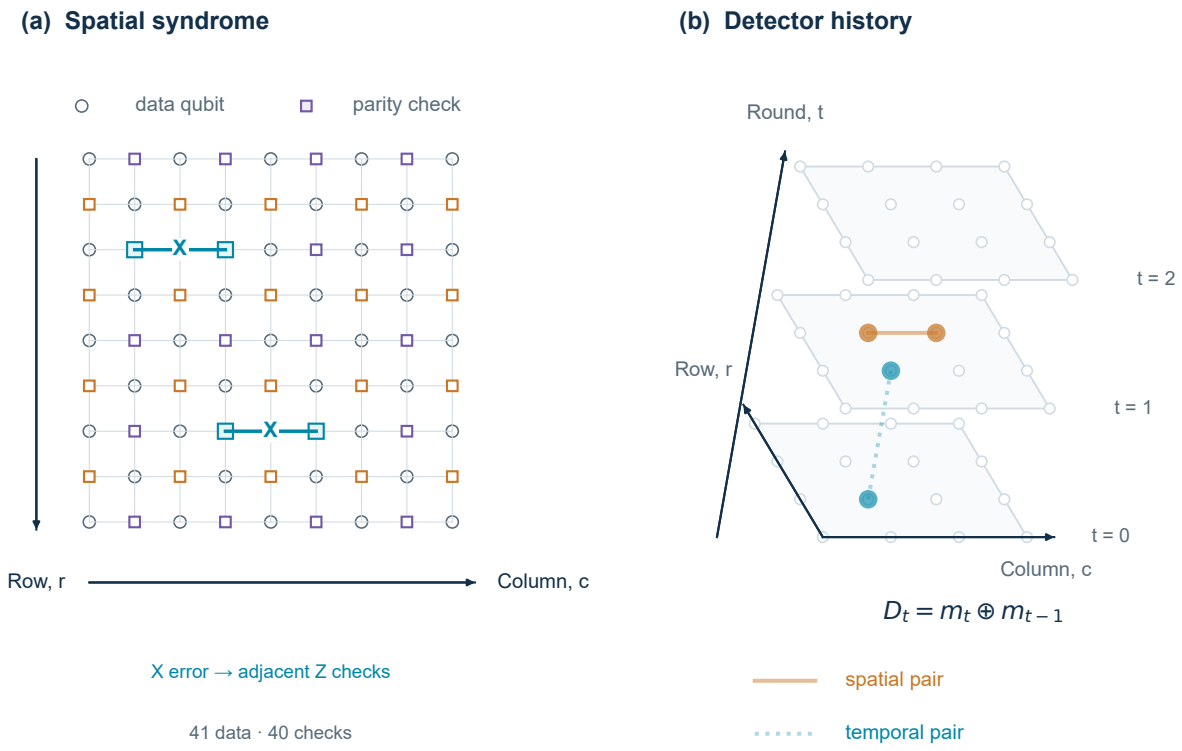


Figure 2.4: Spatial syndromes versus detector histories

(a) Each marked X error fires its two neighbouring Z checks in a single spatial frame. (b) With repeated measurements, detectors form a stack of spatial planes. Two events on nearby checks in the same round suggest a data error. Two events on the same check in successive rounds suggest a measurement error in the reset circuit shown here. The circuits run on the IBM quantum processor use a different comparison because the ancillas are reused without reset (Section 2.3).

Table 2.1: Effect of one incorrect readout on two detector rules. A 1 in either event row means the detector fires. The example starts with a known zero parity and zero reference readings before round one.

Round	1	2	3	4	5
Reported bit	0	0	1	0	0
Detection event with reset	0	0	1	1	0
Detection event without reset	0	0	1	0	1

2.4 Decoders used in this thesis

Minimum-weight perfect matching (MWPM). The detection events are placed on a graph whose edges represent possible single faults, each with a weight reflecting how likely it is. MWPM pairs the events, or connects them to a boundary, so that the total weight is as small as possible. It is a widely used reference for independent-error models. Sparse Blossom finds the matching directly on a sparse decoding graph [9]. The 2D study uses uniform weights for the two error types separately. The 3D study compares *parity-check matrix (PCM) matching*, built from a simple parity-check description, with *detector error model (DEM) matching*, whose graph and weights come from the circuit's noise model.

A weight is a cost assigned to a candidate explanation. If two events can be joined by two ordinary steps or by three equally likely steps, uniform weights favour the shorter route. A circuit fault can instead produce both events in a single step on the circuit-derived graph. A decoder whose graph includes that fault can assign the pair a different cost. The minimum-weight rule stays the same when the graph changes, but it may favour a different explanation.

Correlated matching. The 2D MWPM baseline treats bit and phase errors independently. A single Y error, however, can cause both kinds of check to fire around the same qubit. Correlated matching uses a first match to adjust the weights of edges that could share a physical cause with the selected edges. It then runs matching again [21, 22]. On the 3D circuit graph this method is called *correlated DEM matching*. Chapter 7 compares these choices on the same trials.

Hard-decision renormalisation group (HDRG). HDRG starts with a short search radius and groups nearby fired checks into clusters. A cluster containing an even number of checks can be cleared internally. A cluster that reaches a suitable boundary can send its correction there. Odd clusters that cannot reach a boundary are reconsidered at a larger radius. The implementation follows the method described by Gicev et al. [10, Appendix F] and specifies how to choose a path when several are possible.

Local pre-decoding with global cleanup. The CNN takes the syndrome as its input and proposes a local correction c_L. Every check affected by this correction is toggled, giving the *residual* $s \oplus Hc_L$ from the observed syndrome s. Here H is the parity-check matrix that maps errors to check changes, and $\oplus$ is bitwise exclusive OR. The global decoder proposes a correction c_R for this residual, and the final correction is $c_L \oplus c_R$. Figure 2.5 shows this chain. The aim is to remove short local patterns and so reduce the work left for the global decoder. A local proposal can also introduce new fired checks or change the logical outcome, which is why the complete decoder must be tested.

A controller can usually record Pauli corrections in a classical *Pauli frame* and account for them in later operations or measurements, rather than immediately applying physical gates.

The same three-qubit code illustrates how cleanup can fail. Write the syndrome as an ordered pair $[s_1, s_2]$, where s_1 records the check on qubits 1 and 2 and s_2 the check on qubits 2 and 3. Suppose the syndrome is $[1, 1]$ and the CNN proposes flipping the first qubit. That flip changes only s_1, leaving $[0, 1]$ for the global decoder. If the global decoder now proposes flipping the third qubit, the combined correction clears both checks by flipping both end qubits. Had the original error been the middle flip in Figure 2.1, all three data qubits would now be flipped relative to their initial state. The final syndrome is zero, yet the logical value has changed. The logical outcome therefore determines whether the combined correction succeeds. A count of the remaining checks would not detect this failure.

2.5 The two neural networks

Figure 2.6 shows how both networks map a grid of checks to corrections. Each stack in the diagram holds several values at each grid position.

The 2D network [10] takes four binary input channels on the $(2d-1) \times (2d-1)$ grid: the fired X checks, the fired Z checks, and two fixed masks marking the boundaries. Three convolutional layers with 11×11 kernels map these through $4 \to 9 \to 9 \to 2$ channels. The two outputs at each data position propose an X and a Z correction. The network has 16,335 weights and 20 biases, independent of d. Each layer uses inputs up to five positions away in every direction, so after three layers each output depends on a 31×31 neighbourhood.

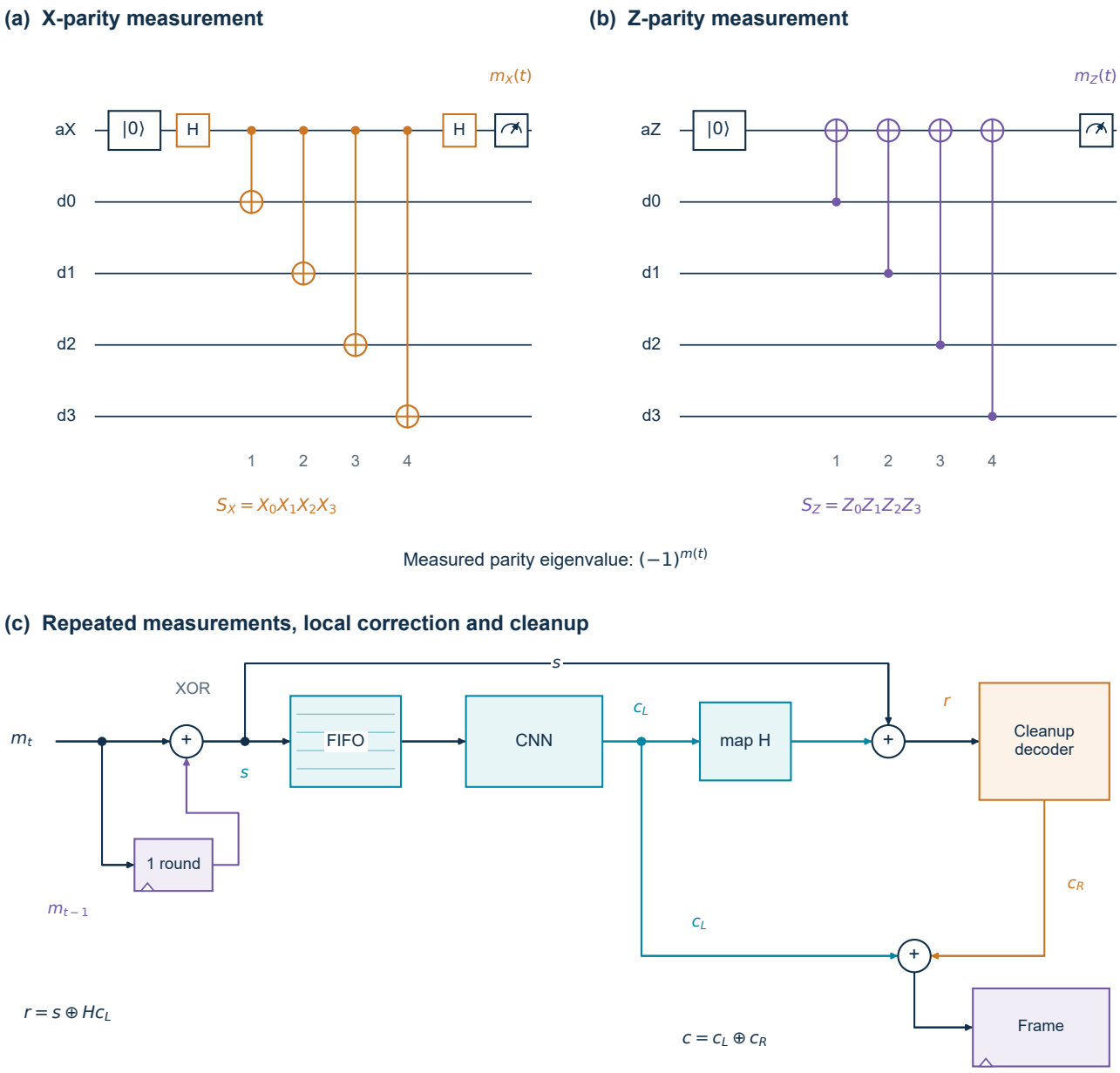


Figure 2.5: Measuring checks and combining local and global corrections

(a, b) Circuits that measure one four-qubit X or Z check into one classical bit. (c) In this reset-based example, consecutive readouts form detection events. The local network proposes c_L. The residual $r = s \oplus Hc_L$ is passed to the cleanup decoder, whose correction c_R is combined with c_L to update the Pauli frame.

A convolutional layer computes, for output channel o at position u,

$$z_o(u) = b_o + \sum_i \sum_\delta w_{o,i,\delta}\, a_i(u+\delta),$$

summing over input channels i and kernel offsets δ. The input value is a_i, the trained weight is $w_{o,i,\delta}$, and b_o is the bias added to output channel o. At each edge, an 11×11 window needs values up to five positions beyond the grid. *Zero padding* supplies zeros for those missing positions. The window moves one position at a time and gives one output at every original position. This choice is called *SAME* padding because the output has the same height and width as the input. Hidden layers apply the rectified linear unit (ReLU), $\max(z, 0)$. The final layer's output z is a *logit*: the original network applies a sigmoid and fires when $\sigma(z) > 0.5$, which is exactly the condition $z > 0$.

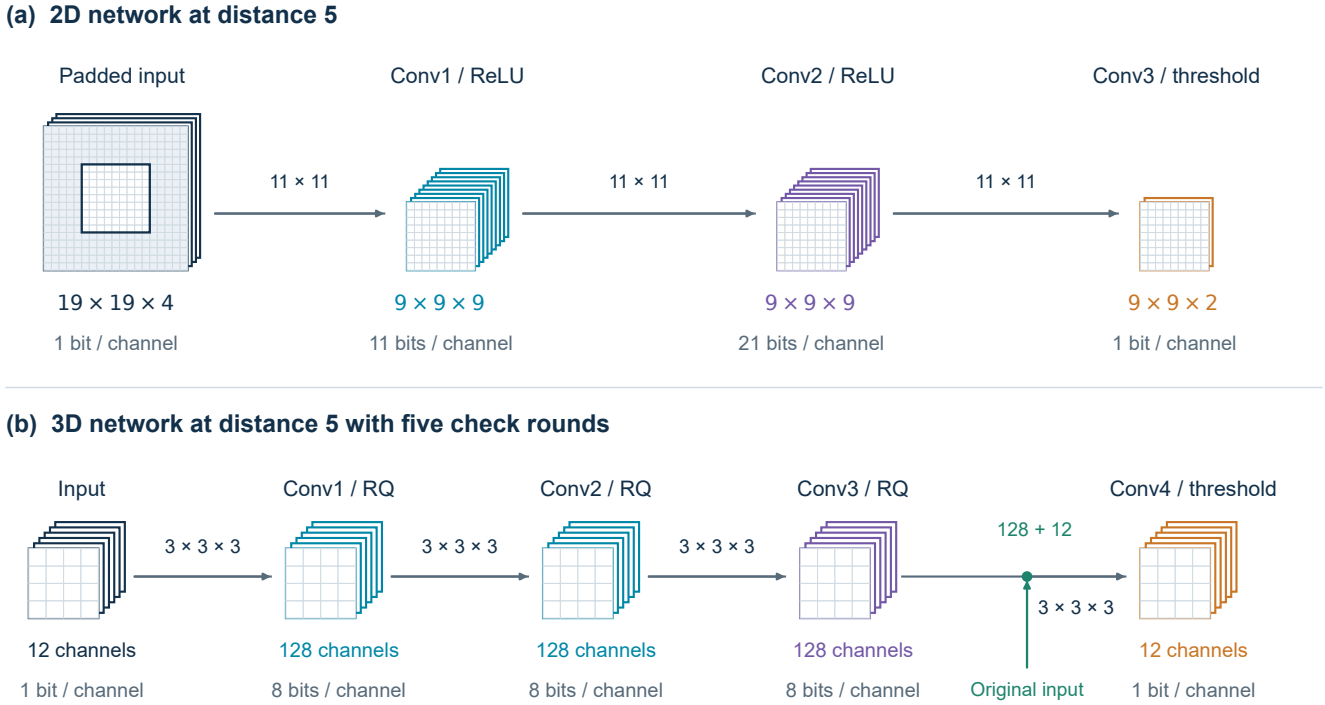


Figure 2.6: The two convolutional decoder networks

(a) The outlined 9×9 distance-5 input has five zero positions added on each edge before the first 11×11 convolution. The hidden maps keep the native spatial size and use 11 and 21 bits per channel. (b) Each stack represents the same six time slices of 4×4 unit cells. The number below each stack gives its channel count. RQ is ReLU followed by requantisation. The green arrow brings the twelve original input channels into the final convolution alongside 128 hidden channels.

A small example makes the calculation concrete. For a window $(1, 0, 1)$, weights $(2, -1, 3)$ and bias -1, the sum is $-1 + 2(1) - 1(0) + 3(1) = 4$, which ReLU keeps. The window $(0, 1, 0)$ instead gives -2, which ReLU replaces by zero. The full network repeats this operation across several channels and two or three axes.

Each stored weight belongs to an offset in the window. When the window moves to the next position, the same weight now multiplies a different input. This is why one set of learned weights can be applied to a larger lattice. Zero padding keeps the array dimensions the same at an edge, but two identical-looking event patterns can have different meanings near an edge and in the interior. The boundary masks give the network that additional information about where the pattern occurs.

A channel is one field on the same grid, analogous to one colour channel of an image. Thus 128 hidden channels mean 128 stored features per position, not 128 physical qubits.

The 3D network [11] processes a space-time volume. Gicev et al. group the rotated code into repeating spatial *unit cells* and stack these cells through time. Each position in this three-dimensional array is a *voxel*, one unit cell at one time slice. For the odd distances studied here, R rounds give $(R + 1) \times \frac{d+3}{2} \times \frac{d+3}{2}$ voxels including the final data readout. At $d = R = 5$ this is $6 \times 4 \times 4 = 96$. Each voxel has twelve binary channels: eight *presence* channels that mark which sites exist at that position of a repeating unit cell, and four detector channels. Three hidden layers of $3 \times 3 \times 3$ convolutions map $12 \to 128 \to 128 \to 128$ channels. The original twelve inputs are then concatenated with the 128 hidden channels, and a final convolution maps these 140 channels to twelve outputs: four X corrections, four Z corrections and four proposed measurement-error corrections.

The network has 971,568 weights and 396 biases, about sixty times more than the 2D network. Its four layers give each output a $9 \times 9 \times 9$ neighbourhood, so a decision about round t needs data up to round $t + 4$.

The eight presence channels distinguish an absent site from a site where no detector fired. They stay fixed across trials on the same patch. The hidden channels contain learned features instead. Storing 128 values per position makes the hidden maps much larger than the input, even though they describe the same patch. This is one reason the two networks pose different hardware problems. In 2D the challenge is supplying the arithmetic with data quickly enough. In 3D the first challenge is fitting the weights in memory.

2.6 FPGA essentials

An FPGA is built from a few kinds of configurable resources [23, 24, 25], shown in Figure 2.7:

- **lookup tables (LUTs)**, programmable truth tables that implement logic or small memories;
- **flip-flops (FFs)**, one-bit registers that hold values between clock edges;
- **block RAM (BRAM)**, larger memories that each hold 36,864 bits (36 Kibit) with a limited number of read and write ports; and
- **digital signal processing (DSP) blocks**, dedicated multiply-accumulate units (here a 25×18-bit multiplier with a 48-bit accumulator).

The XC7A100T on the Arty board has 63,400 LUTs, 126,800 FFs, 240 DSP blocks and 135 BRAM36 blocks, or 4.98 Mbit of block memory [14, 26]. Two of these numbers recur throughout the thesis: 240 multipliers must be reused to compute many thousands of products at each grid position, and 4.98 Mbit is less than the 3D network's weights need at eight bits.

Parallelism and memory. An FPGA design can trade hardware for time. One multiplier reused over many clock cycles takes little space but more time. Many multipliers can work in parallel, but need more hardware. Additional arithmetic is useful only if the memories can supply its operands. A memory *word* is the group of bits read from one address. A true dual-port BRAM can read two addresses in one cycle, one on each port. To read more addresses in the same cycle, the design can copy the data into several memories or divide the array into *banks* so the required values occupy different memories. Suppose four lanes need four different values from an array. Four values packed into a single wide word can be read together if they always share an address. If the lanes need four independent addresses, a single two-port memory cannot serve all four in the same cycle. Banking helps when the requests address different banks. If they all address the same bank, some must still wait. The address pattern is therefore as important as the number of stored bits. Both kinds of access occur in the designs of Sections 5.3 and 6.4.

All these circuits can share a clock and still perform several calculations per cycle. Suppose an output is the sum of eight products. One multiplier can form the eight products over eight cycles, before allowing for pipeline delays. Four multipliers can form four terms together and finish issuing the products in two cycles. Alternatively, four copies of the original one-multiplier engine can work on four different outputs in parallel. These are different uses of four lanes. The first shortens one sum, while the second computes several sums concurrently. Both use the same clock, need

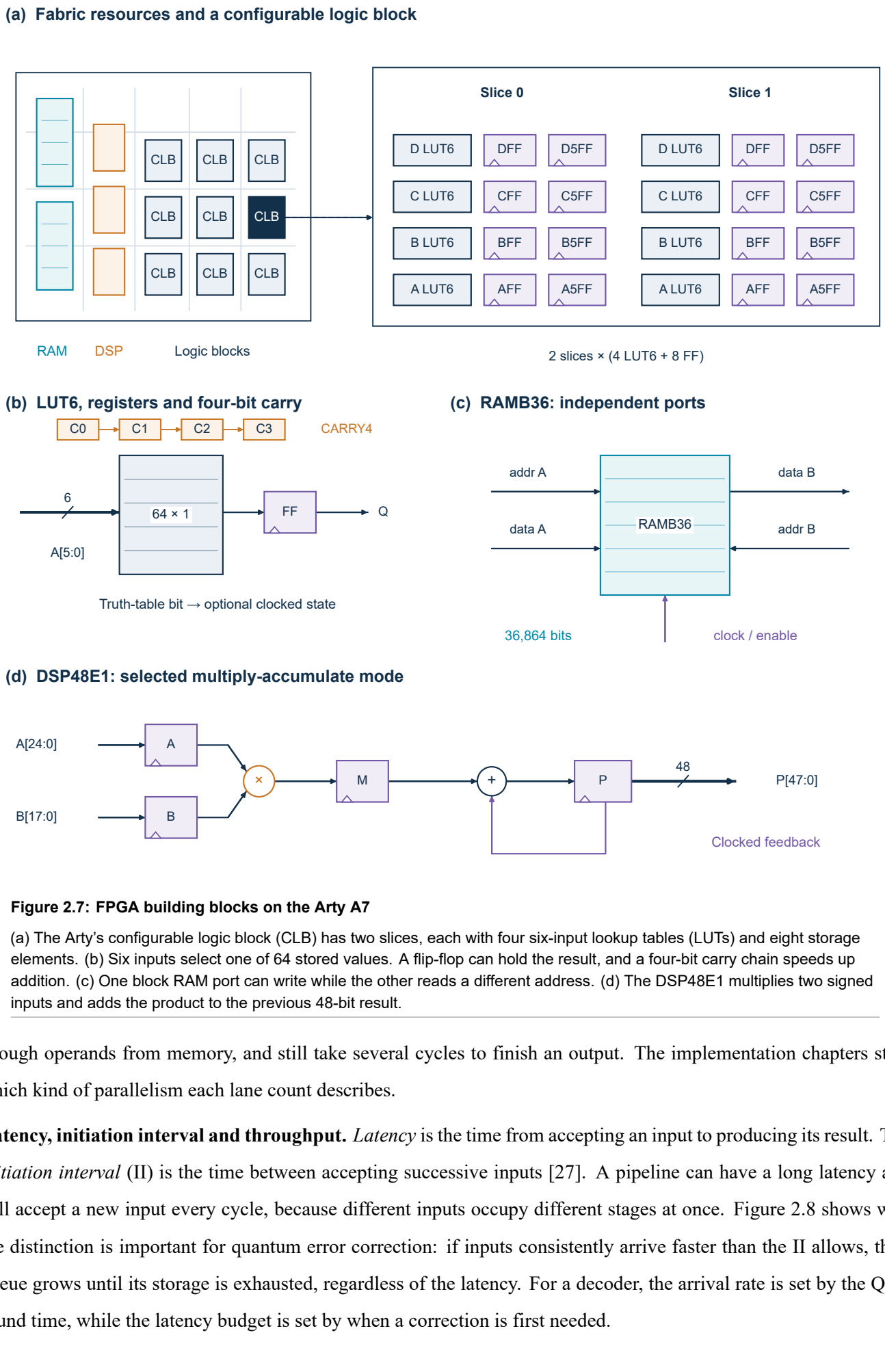


Figure 2.7: FPGA building blocks on the Arty A7

(a) The Arty's configurable logic block (CLB) has two slices, each with four six-input lookup tables (LUTs) and eight storage elements. (b) Six inputs select one of 64 stored values. A flip-flop can hold the result, and a four-bit carry chain speeds up addition. (c) One block RAM port can write while the other reads a different address. (d) The DSP48E1 multiplies two signed inputs and adds the product to the previous 48-bit result.

enough operands from memory, and still take several cycles to finish an output. The implementation chapters state which kind of parallelism each lane count describes.

Latency, initiation interval and throughput. *Latency* is the time from accepting an input to producing its result. The *initiation interval* (II) is the time between accepting successive inputs [27]. A pipeline can have a long latency and still accept a new input every cycle, because different inputs occupy different stages at once. Figure 2.8 shows why the distinction is important for quantum error correction: if inputs consistently arrive faster than the II allows, their queue grows until its storage is exhausted, regardless of the latency. For a decoder, the arrival rate is set by the QEC round time, while the latency budget is set by when a correction is first needed.

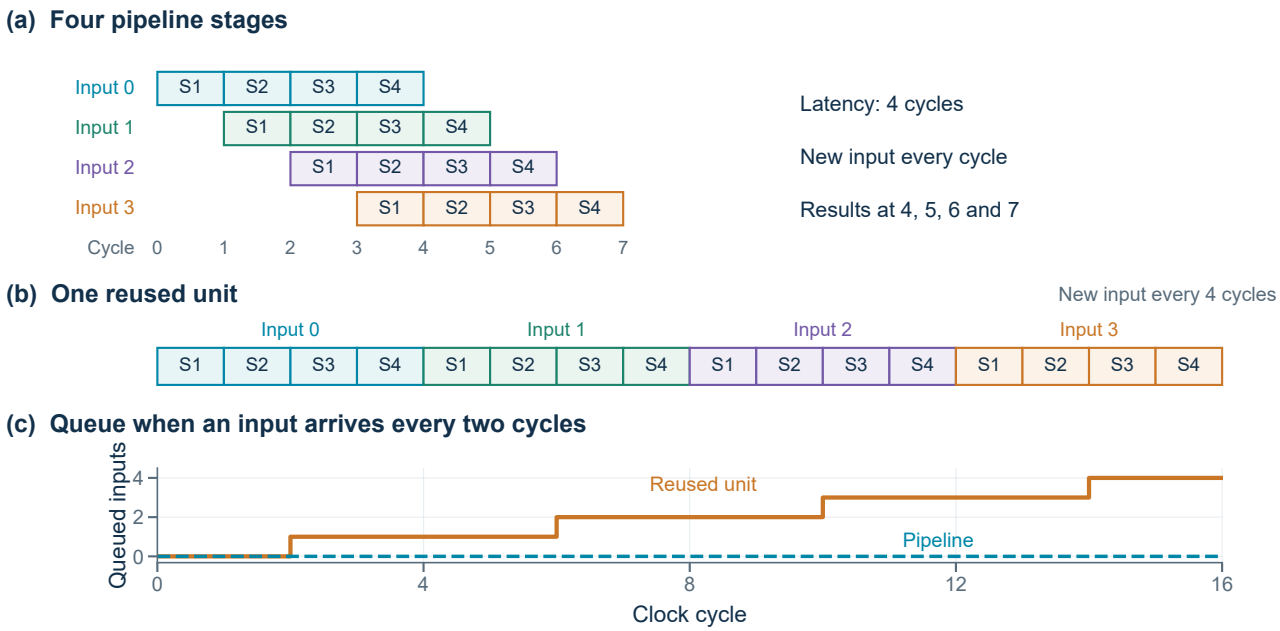


Figure 2.8: Latency, initiation interval and waiting inputs

(a) Each input passes through four stages. Results emerge four cycles after acceptance and one cycle apart. (b) The shared unit takes four cycles per result and processes one input at a time. (c) With arrivals every two cycles, the shared unit has four waiting inputs at cycle 16. The count is taken after arrivals and dispatch and excludes the input in service.

Timing closure. A design is described at the *register-transfer level* (RTL) in a language such as Verilog, which defines registers and the logic between them. RTL simulation checks values cycle by cycle. Synthesis maps the logic to FPGA resources, and placement and routing determine their physical locations and connections. Static timing analysis then reports *slack*: the margin by which signals satisfy their setup and hold requirements around a clock edge. For a register to capture a value reliably, the value must settle a short time before the clock edge. Negative setup slack means that, under the timing conditions analysed, a path can deliver the value after this deadline. The receiving register could then capture an incorrect value, even when a short board test succeeds. In this thesis, a timing-qualified clock passes the routed timing checks under the stated constraints. A future quantum-controller interface needs its own input/output constraints and checks. Halving the cycle count halves the run time only if the clock frequency is maintained.

Clock domains. A design can contain several clocks, for example one for an interface and another for the core, or separate clocks on different boards. A signal passing between them may be sampled while changing. Such *clock-domain crossings* need dedicated synchronisation. A multi-bit stream often uses an asynchronous first-in, first-out (FIFO) queue, whose read and write sides run on different clocks. Chapter 8 uses these queues between distributed workers. The Arty's serial receiver also synchronises its asynchronous input before decoding incoming bytes.

A common digital interface uses *ready* and *valid* signals. A record transfers on a clock edge only when both are asserted. If the sender has valid data but the receiver is not ready, the sender holds that same record unchanged until it is accepted. This is *backpressure*. A FIFO can absorb a short stall, but it cannot compensate for a sustained mismatch between arrival and processing rates. A running quantum experiment also cannot be assumed to pause whenever a decoder queue fills, which is why the system needs a stated buffer capacity and a response to overload.

The next chapter examines how earlier decoders addressed these requirements and where their measurements can be compared with those in this thesis.

3. Related work and the research gap

Comparisons between published decoders must account for their input data and the interval over which latency was measured. A single pattern of perfect checks is a different input from a history of noisy measurements. Likewise, a time measured at a network's output covers less work than one measured when the quantum controller receives a correction. This chapter reviews decoding algorithms and their hardware implementations to identify the gap addressed by this thesis.

3.1 Decoding speed and resource costs

A decoder that consistently processes fewer rounds than arrive accumulates a backlog. Some operations must wait for its answer, and continued syndrome generation during those waits can cause severe, potentially exponential, growth in program runtime [6, 28]. Hardware cost is a further constraint. Maurya and Tannu propose sharing a smaller pool of hardware decoders among logical qubits [28]. Barber et al. likewise emphasise resource efficiency [29]. The decoder therefore needs enough throughput to process arriving rounds and low enough latency to return each correction when it is needed.

3.2 Global decoders in previous work

The choice of matching implementation affects both the speed and the accuracy of the baseline. PyMatching made MWPM practical for large simulations [30]. Its Sparse Blossom algorithm works directly on a sparse detector graph. Higgott and Gidney report less than a microsecond of processing per round on one CPU core for a distance-17 circuit with 0.1% depolarising noise [9]. The quality of the correction still depends on how accurately the graph represents physical faults [6].

A timing comparison with a CNN must account for the workload. A round at low noise can contain very few events, and the search performed by a matching decoder depends on which events fire. The convolutional core in this thesis visits every scheduled position even when its input is all zeros. A matching time measured at one noise level therefore cannot be added directly to a CNN time measured on another workload. To measure the combined cost, the second stage must be timed on the residual left by the first (Section 8.5).

Fowler showed how a first matching can identify likely correlations and inform a second matching [21]. PyMatching implements this approach for correlated error models [22]. Correlated matching tests whether the CNN still helps when the global decoder uses some of the same correlations.

Cluster decoders offer a different baseline. Delfosse and Nickerson's union-find decoder has almost-linear running time in the size of the code [31]. Gicev et al. used HDRG to clean up their CNN's output [10, Appendix F]. This thesis uses HDRG to compare with that work and MWPM to test how the choice of cleanup algorithm changes the result.

3.3 Neural decoders

Early neural decoders. Torlai and Melko showed that a neural network could learn to decode small topological codes from examples [32]. Varsamopoulos, Criger and Bertels decoded small surface codes with feed-forward networks [33],

and Chamberland and Ronagh trained deep networks for small fault-tolerant experiments with circuit noise [34]. These studies established learned decoding as a practical approach for small codes. Their demonstrated models use fixed input and output layouts, so scaling requires attention to both network structure and training cost.

The 2D fully convolutional decoder. Gicev et al. introduced the network implemented in Chapter 5 [10]. Its number of parameters does not depend on code size: the same trained kernels apply across distances. It was trained on approximately fifty million distance-33 examples, including patches with defects, using staged noise levels [10, Sec. 2.3]. The simulation applied independent depolarising errors to the data qubits and assumed perfect syndrome extraction. Faults in the measurement circuit were therefore excluded. Under these conditions, five CNN passes followed by HDRG gave a threshold near 13.8%, compared with 11.5% for HDRG alone [10, Sec. 2.4, Fig. 5]. The paper also compares one-pass runtime with an earlier PyMatching version [10, Fig. 6].

A threshold of 13.8% is a statement about the noise at which increasing the distance stops improving protection. It does not mean that 13.8% of the decoded examples failed. For comparison with an FPGA implementation, the failure rate can be measured at a specific operating point, such as distance 33 at 13% physical error probability. It can then be considered alongside the inference time for that patch. Chapter 7 uses operating points of this kind to compare the integer network with current cleanup decoders.

The 3D space-time classifier. The space-time model extends the approach to repeated noisy measurements [11, 35]. The classifier implemented here was trained on the original, unsimplified correction labels in arXiv version 1. It has three hidden layers of 128 channels and was trained on about 500,000 distance-33 examples with circuit-noise parameter p' between 0.001 and 0.007 [11, Sec. 2.3.2]. The authors report improvements over PCM matching but generally not over DEM matching. Their iterative diffusion decoder is a separately trained network. The repeated-pass experiments in this thesis reuse the classifier and its original weights.

The same group has also used repeated syndrome measurements on IBM quantum processors to show that device data are inconsistent with uniform depolarising noise and contain significant temporal correlations between detection events [36]. This is one reason the decoders were tested on device records as well as in simulation.

Other neural approaches. Meinerz, Park and Trebst combine a sliding local network with a global decoder on toric codes [37]. Chamberland et al. study local 3D convolutional pre-decoders under circuit noise and show that the treatment of the residual is itself a design problem [17]. AlphaQubit predicts logical outcomes directly with a recurrent transformer and achieves high accuracy on experimental data [38]. Its larger network has different hardware requirements from the local CNNs studied here. Overwater, Babaie and Sebastiano reported that fully connected neural decoders for distance 3 and distance 5 fitted on an Artix-7 FPGA. Their FPGA timing figures are post-implementation estimates. They also estimated ASIC costs [39].

Zhang et al. built a decoder with 3D convolutional layers followed by fully connected layers. They tested its FPGA and control electronics with waveforms generated from simulated syndrome bits. The paper reports decoder times of 197 ns at distance 5 and 1.136 µs at distance 7, using ten and fourteen measurement rounds respectively [40, Sec. 3.2,

Table 3]. Their decoder returns a correction for the patch. The CNNs studied here predict local corrections and leave unresolved checks to a second decoder.

Yan et al. compared five neural architectures through distance 9 and used high-level synthesis to estimate the cost of a four-bit temporal CNN on the VP1902 [41]. More recently, Chamberland et al. demonstrated CNN pre-decoders that improve both logical error rates and total decoding time when combined with PyMatching [42]. Section 7.4 discusses how these results qualify the conclusions for the networks studied here.

3.4 Neural-network inference on FPGAs

Outside quantum computing, neural-network inference on FPGAs is a mature field, and two of its ideas are used in this thesis. The first is *integer-only inference*. Jacob et al. showed how a trained network can be executed entirely in integer arithmetic: weights and activations are stored as small integers with a scale factor, accumulation is exact, and each layer's result is rescaled to the activation range by a fixed-point multiplier and a shift [43]. The 3D requantiser follows this pattern, with per-channel multipliers and per-layer shifts. The smaller 2D network can keep its activations at full width (Section 4.2).

The second idea is *dataflow* acceleration: different layers work concurrently, passing results to the next layer when they are ready. FINN [44] and hls4ml [45] generate FPGA implementations with adjustable arithmetic parallelism and reuse. hls4ml was developed for particle-physics triggers with sub-microsecond inference requirements, while FINN emphasises efficiently quantised networks. For a CNN, adjacent layers must be scheduled around the availability of their inputs. A neuron in the second layer needs a small window from the first layer. As soon as every value in that window is available, the second layer can start producing its first result while the first layer moves on. If a required value is still missing, the second layer must wait. Memory layout and the time at which each value becomes available are therefore as important as the number of multipliers.

The designs use hand-written RTL so that the integer formats, memory layout and layer schedules are controlled explicitly. This allowed the same arithmetic reference to be checked across several resource-constrained implementations.

3.5 Real-time decoding in hardware

A decoder can process rounds as quickly as they arrive even if the final correction takes many rounds to produce. Google Quantum AI demonstrated this in a distance-5 memory on a superconducting processor. Their streaming software decoder sustained the arrival rate of 1.1 µs rounds in experiments lasting up to a million rounds. Table 3.1 gives the delay from the last syndrome to the correction [8].

Barber et al. tested FPGA collision clustering and developed an ASIC physical design for the same algorithm. On simulated finite memories they report megahertz decoding rates up to 881 qubits on the FPGA and 1,057 for the ASIC design [29]. Caune et al. integrated an FPGA cluster decoder into Rigetti's control system. Their nine-round stability experiment timed the full response, including communication, decoding and the conditional operation [13]. Yang et al. connected a recurrent neural decoder on a Kintex-7 FPGA to a distance-3 superconducting memory [46]. Both experiments timed the feedback path to a physical processor.

The Quantum Instrumentation Control Kit (QICK) uses a radio-frequency system-on-chip (RFSoC) to bring qubit readout and control onto the same device [12]. Other qubit technologies operate on very different time scales (Section 8.3). Trapped-ion experiments report complete adaptive cycles below 200 ms [47], and neutral-atom architectures combine slower extraction with atom loss and deferred readout [48, 49]. A decoder's speed must therefore be considered in the context of the quantum system it serves.

Table 3.1 places those results beside selected decoding experiments. In particular, a controller response includes work after the decoder has produced its result. A timer that stops at the CNN output also excludes the global cleanup.

Table 3.1: Timing boundaries in published decoder experiments and the spatial CNN tests.

Experiment	Reported time	Interval timed
Google Quantum AI, $d = 5$ [8]	63 ± 17 µs	Last syndrome at the software decoder to the returned correction
Caune et al., nine rounds [13]	9.6 µs	Readout to conditional operation, including 6.5 µs for clustering
Zhang et al., $d = 5, 7$ [40]	197 ns, 1.136 µs	FPGA decoder on ten or fourteen rounds of synthetic input
Yang et al., $d = 3$ [46]	124 ns, 550 ns	Neural inference alone, then the full readout-to-feedback path
This thesis, Arty, $d = 5, 33$	38.73 µs, 1.154 ms	Loaded spatial frame to final CNN prediction, measured on the board
This thesis, VP1902, $d = 33$	21.8 µs	Same CNN interval, 2,180 simulated cycles at a routed 100 MHz

3.6 Parallel and distributed decoding

Distributed decoders can divide different parts of the calculation. A convolution can be split *spatially*: each worker computes its own region of the output using a border of neighbouring inputs, called a *halo*, and the tiles combine to reproduce the undivided output exactly. A measurement history can be split into overlapping *time windows*, with rules that determine which window commits a correction near a boundary [50, 51]. A global matching problem can be split and *fused*, as in Fusion Blossom [52]. Varsamopoulos et al. evaluated distributed neural decoding numerically [53].

Sections 8.6 and 8.8 test the first approach. For example, a correction assigned to the last column of one spatial tile may depend on a check in the first column of the next tile. The worker needs a copy of that check, even though its neighbour owns the output at that position. Sending that input lets both workers use the original convolution at the tile boundary. After the tiles have produced their predictions, the global decoder must still resolve any events left across that boundary.

3.7 The research gap

The 2D and 3D CNNs reuse their trained weights as distance increases. Their original papers report accuracy and software runtime [10, 11]. The earlier neural FPGA studies above cover distances 3 and 5 for the fully connected decoder, 5 and 7 for the convolutional decoder and 3 for the recurrent decoder. The project proposal asked how much of the local CNN computation a small FPGA could perform as the patch grew [54]. The spatial network ran on the

Arty through distance 33 (Table 5.1). The larger patch requires more arithmetic and memory accesses, although the weights occupy the same storage.

The proposal expected approximately constant-time inference as the lattice grows. Achieving that would require parallel hardware to grow with the lattice. A fixed small FPGA instead reuses its arithmetic, so identifying its resource limits became part of the contribution. The models were tested with their trained weights, including the effects of integer precision, the output threshold and removal of small weights.

Comparisons must also account for the number of CNN passes. Compared with one pass, running all five requires four additional CNN evaluations and residual updates. A failure rate from five passes does not show what a single hardware pass achieves (Section 7.1.2). Chapter 4 sets out how numerical correctness, inference time and decoding quality were tested.

4. Method: from a trained network to verified hardware

The hardware tests compare each decoder core with a reference that uses the same integer arithmetic. A separate question is whether the resulting corrections preserve the encoded qubit.

4.1 Overview

The hardware workflow begins by converting trained floating-point weights to integers. A Python *integer reference* computes every layer as the hardware should, and writes *golden vectors*: inputs together with the expected raw sums, activations and decisions at every layer. RTL simulation compares the hardware description against those vectors, value by value. Synthesis and routing in Vivado then map the design onto the FPGA and report resources and timing. Finally, the board receives inputs over a serial link, runs the core and returns its decisions and a cycle count. The host checks the decisions against the integer reference.

Two comparisons locate numerical differences. If the hardware and integer reference disagree on the same input, the hardware calculation must be checked against the reference. If the integer and floating-point references disagree, converting the network to integers has changed its prediction. A separate experiment in Chapter 7 tests whether the encoded qubit is preserved after cleanup.

4.2 Integer arithmetic for the 2D network

A trained weight w is represented by an integer $q_w = \text{round}(w/s_w)$ and a scale s_w, so that $w \approx s_w q_w$. The design uses signed eight-bit weights with one scale per layer. Each input to the first layer is either zero or one. The hardware therefore adds zero or the corresponding weight without using a multiplier.

Here the scale gives the change in real value represented by a change of one in the stored integer. The representable weights are evenly spaced. A weight between two of them is rounded to the nearer value. The FPGA and the integer reference both receive the same resulting integers. Agreement between those two implementations can be exact even when their predictions differ from the original trained network.

The bias must use the same units as the products it is added to. If activations have scale s_a and weights scale s_w, their products have scale $s_a s_w$. The biases in each layer have a separate scale s_b. Before inference, each quantised bias q_b is converted into the units of its layer's accumulator. The binary input has scale one. After two convolutions, the layer-two sum has scale $S_2 = s_{w1} s_{w2}$, the product of the two layers' weight scales. Its stored bias is $B_2 = \text{round}(q_b s_b / S_2)$. An eight-bit quantised bias therefore does not imply an eight-bit hardware bias. The exported values are 48-bit integers in accumulator units, while each hardware bias memory uses its layer's accumulator width. Figure 4.1 shows the resulting datapath for one output channel.

For example, an activation of 1.5 with scale 0.25 is stored as integer 6. A weight of 2.5 with scale 0.125 is stored as 20. Their integer product, 120, has scale $0.25 \times 0.125 = 0.03125$ and therefore represents 3.75. Adding a real bias of 0.5 requires sixteen of those accumulator units, giving integer 136, or 4.25 in real units. Adding integer one would add only 0.03125. The same conversion is applied to the trained coefficients.

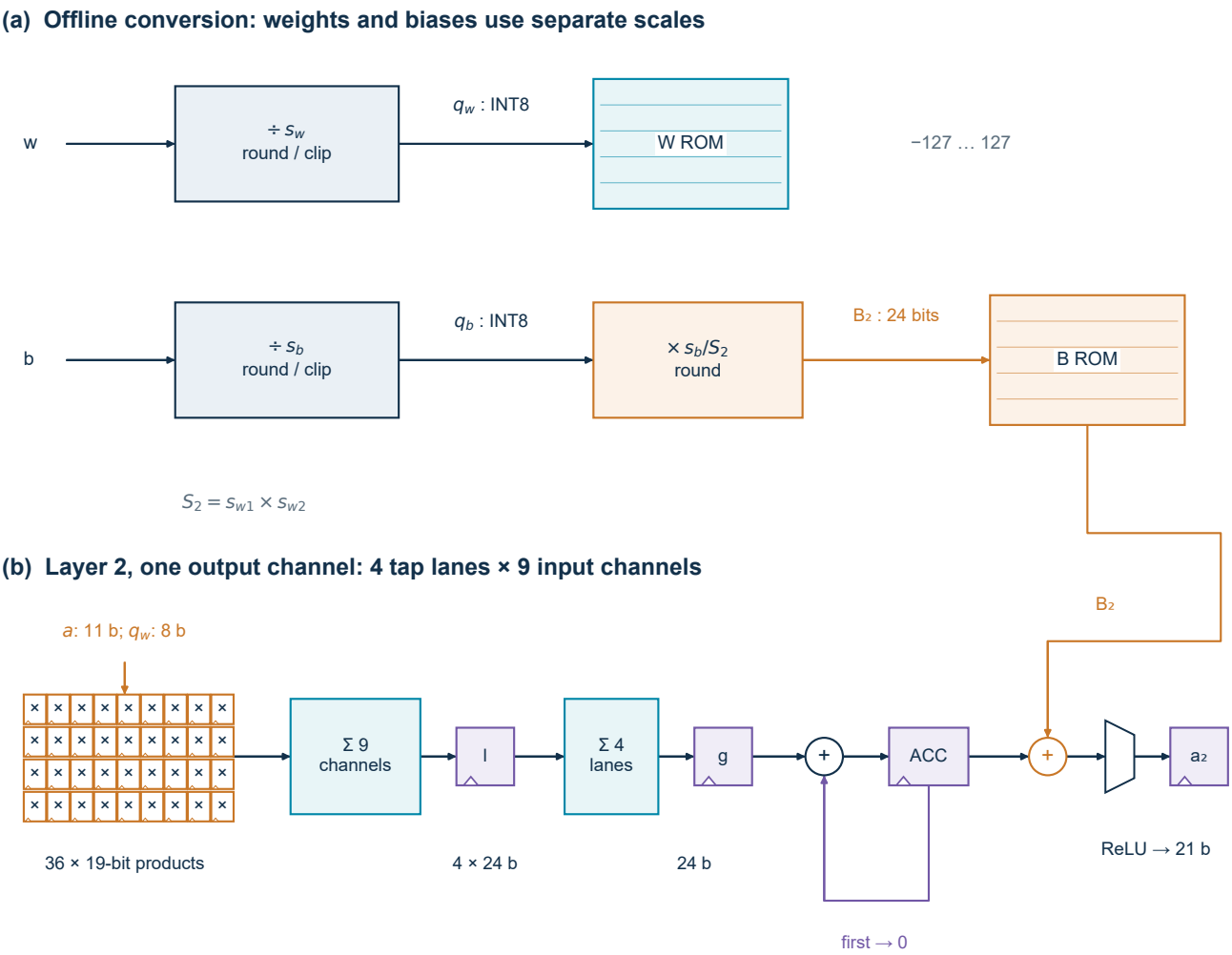


Figure 4.1: Integer scales and the layer-two datapath in the 2D network

(a) Weights and biases are quantised separately to eight-bit integers. Before the FPGA runs, the layer-two biases are converted to the same units as the products and loaded into its 24-bit bias memory. (b) For one output channel, each of four tap lanes forms nine signed 11 × 8-bit products. Two stages sum the 19-bit products into a 24-bit accumulator. After the last group, the circuit adds the bias once and applies ReLU to produce a 21-bit activation.

There is no rescaling between layers. Hidden activations are kept at full precision, which makes them wider (11 and 21 bits after layers one and two) but avoids intermediate rounding after the coefficients have been converted. The final decision uses the strict test $z > 0$, which is equivalent to $\sigma(z) > 0.5$ for the same logit, so the circuit does not need to compute a sigmoid. The strict inequality is required: a test of the sign bit alone would accept $z = 0$, and integer conversion can produce exact zeros. A check over 500 generated frames, 4,225,000 decisions, found no disagreement between thresholding the activation and applying the strict-positive test. Appendix A.2 gives the bounds and the decision-rule check.

4.3 Integer arithmetic for the 3D network

The 3D network needs a different approach, because its weights must be compressed to fit in memory (Section 6.3). Each output channel has its own weight scale, with symmetric integers and no zero offset. Layers one and four use

eight-bit weights and the two large middle layers use four-bit weights. Hidden activations are unsigned eight-bit values, so after each hidden layer the 32-bit accumulator A is *requantised*:

$$a = \min\left(255, \left\lfloor \frac{\max(A, 0)\, M + 2^{S-1}}{2^S} \right\rfloor\right),$$

where M is a positive per-channel multiplier and S a per-layer shift (15, 20 and 19 for the three hidden layers). The added 2^{S-1} rounds non-negative values to the nearest integer, with halfway cases rounded up. The minimum limits the stored activation to 255.

The skip connection brings the binary input into the last layer at a different scale, so its weights are stored as sixteen-bit values already converted to the last layer's accumulator units. Each bias also includes a correction for the average error introduced by quantisation. Calibration used 128 synthetic distance-5 X-memory inputs, each with six time slices at $p' = 0.005$, to select activation ranges and these bias corrections. It fixes one integer representation of the trained network without retraining it. The evaluations used other inputs.

For an illustrative accumulator $A = 13$, multiplier $M = 3$ and shift $S = 2$, the result is $\lfloor (39 + 2)/4 \rfloor = 10$. A negative accumulator becomes zero through ReLU. With $A = 400$, the same calculation gives 300 before the final minimum clamps it to 255, the largest unsigned eight-bit value. Clamping is different from discarding the excess bits, which would wrap 300 to 44. Specifying the order of ReLU, multiplication, rounding and clamping lets software and hardware agree even at these boundaries.

A separate scale for each output channel matters when channels have different ranges. Suppose two channels need weights up to 1 and 0.1 in magnitude. If four-bit weights share a scale that maps 1 to the integer 7, then every positive weight in the second channel rounds to either zero or one. Its own scale could map 0.1 to 7, leaving several distinct values for that channel. Per-channel scaling preserves this extra detail with the same number of stored bits per weight. The cost is that the hardware must use the scale belonging to the channel it has just computed.

A small change in a network's final output can flip a correction bit if the value crosses zero. Each decoding comparison identifies the precision of its reference (Sections 7.1.3 and 7.2). The 2D floating-point reference stores 32-bit coefficients and uses 64-bit arithmetic. The 3D floating-point reference uses 32-bit arithmetic. The integer references reproduce the weights, biases and scaling used in the FPGA.

4.4 Bit-width bounds

Overflow can change the sign of a sum and reverse a decision. For the fixed coefficients and input formats used here, the bounds below cover every possible input. For a dot product $b + \sum_i w_i x_i$ with $0 \le x_i \le U_i$, the largest possible value takes every positive weight with its input at the maximum, and the smallest takes every negative weight:

$$a_{\min} = b + \sum_{w_i<0} w_i U_i, \qquad a_{\max} = b + \sum_{w_i>0} w_i U_i.$$

For the first layer $U_i = 1$; later layers use the previous layer's bound. The bound is conservative, because the extreme combinations may never occur together, but it does not depend on any assumption about which errors are likely.

For example, consider a first-layer neuron with bias 1 and three binary inputs weighted by $2, -5, 3$. Its largest sum is 6, when only the two inputs with positive weights are active. Its smallest sum is -4, when only the input with weight -5 is active. The all-zero and all-one patterns both give a sum of 1. Those two patterns alone would therefore underestimate the width needed. The bound covers inputs that an ordinary set of test cases might miss.

Signedness and intermediate width both matter. Four-bit two's-complement arithmetic represents -8 to 7. Although 6 and 5 each fit, their sum 11 does not: truncating it to four bits would interpret the result as -5. Similarly, the terms $7, 7, -7, -7$ sum to zero, but a circuit that adds the first pair together must first represent 14. Each partial sum therefore needs a sufficient width as well. The bounds cover the partial sums formed by each tested arrangement of parallel lanes.

For the 2D network the final sums lie between $-3{,}843$ and 638 in layer one, $-987{,}422$ and 543,106 in layer two, and $-2{,}120{,}775{,}324$ and 909,729,481 in layer three, needing 13, 21 and 32 signed bits. The hardware uses 16-, 24- and 36-bit accumulators. The check also covers partial sums for tap groupings from one to eleven lanes, because an intermediate sum can be larger than the final one before positive and negative terms cancel.

For the 3D network every non-negative hidden accumulator entering requantisation stays below 2^{25}, every multiply-and-round value below 2^{47}, and the output including the skip path below 2^{31} in magnitude. The chosen 25-bit requantiser input, 48-bit multiply-and-round path and 32-bit output accumulator are therefore sufficient. Appendix A gives the layer-by-layer bounds in Sections A.2 and A.3. Figure A.1 compares the 2D bounds with the allocated widths and the ranges observed on four test inputs. An independent calculation from the packed coefficient files confirmed the bounds.

4.5 Hardware verification

Golden vectors at every layer. The testbenches compare every raw sum, every stored activation and every decision with the integer reference, and check that outputs appear in the correct order. They also check that a layer never reads a value that the previous layer has not yet written. Test inputs include all-zero and all-one frames, isolated impulses, boundary patterns, sparse and dense random patterns, and saturating inputs. These patterns test arithmetic and control. Chapter 7 evaluates decoding quality on sampled noise and quantum-device measurements.

Independent checks. A separate NumPy implementation reads the packed hardware coefficient files directly and reproduces the golden vectors at every layer of both networks. Icarus Verilog also checks the 2D hardware description on separate inputs at distances 5, 9, 17 and 33. Every decision matches the independent reference, and the cycle counts agree with the corresponding board configurations (Section 5.6). Appendix D.1 states the scope of these additional checks, including the distance-33 arithmetic-resource mapping and which tests inspect intermediate values.

On the board. A UART wrapper receives a complete input frame, starts the core, and returns its cycle counter followed by the packed decisions. The host compares every decision with the reference. The main repeated-transaction campaign covers seven 2D and two 3D configurations, each with 1,000 transactions drawn from 67 distinct inputs. The tests of individual design changes use smaller input sets, described in Chapters 5 and 6. Repeated inputs check that the board gives consistent answers. Appendix B.1 describes the serial interface and the timing of a complete transaction.

Checks of the whole path. The tests covered the complete transaction as well as the arithmetic. Table 4.1 gives four faults found during implementation and the checks that exposed them. Comparing intermediate values located numerical errors, while sending different inputs consecutively exposed control errors that repeating one input could miss. Resource and timing reports then checked whether a numerically correct design could fit and run at the target clock.

Table 4.1: Representative faults found by numerical and complete-transaction tests.

Fault	Check that exposed it	Correction
Coefficients failed to load, leaving unknown values or missing arithmetic	Loading messages, unknown-value checks and resource counts	Include the coefficient files in simulation and synthesis, then check their contents
Negative products changed sign	Compare raw layer sums with the integer reference	Make the product and its extension explicitly signed
Returned decisions were shifted by one position	Simulate the complete serial transaction	Wait for the registered memory address and synchronous read to finish
A new transaction returned the previous frame	Alternate different input frames	Wait for the core's busy acknowledgement before checking completion

An incorrect final bit can originate several layers earlier. Comparing outputs in layer order finds the first mismatch. A correct sum followed by an incorrect stored value indicated a fault in rounding, clamping or a memory write. When the sum itself was wrong, the first checks were the input address, the coefficient and the signed multiplication.

Simple inputs made some faults easier to find. A test with one nonzero input site makes its address easy to follow through the trace. In the first layer, changing one input from zero to one changes an affected raw sum by exactly the corresponding weight. If the change appears one column away, the address or output order is a more likely cause than the arithmetic. An all-zero input tests the biases, which can produce nonzero outputs even when every input is zero.

Alternating two different frames in a board test serves a similar purpose. If both frames produce the same decisions, returning the first frame twice appears correct. If their expected decisions differ, the first decision bit on which they differ is enough to expose a stale result on the second transaction. The cycle counter alone would not detect a correct-length transaction that returned old data.

4.6 Timing and resource measurement

The core cycle counter starts when the core accepts its start command and stops when the last decision has been written. With C cycles at clock frequency f, the core time is C/f. The counter excludes loading the input over the serial link, returning the decisions, global cleanup and any interaction with a quantum controller. Because the schedules contain no data-dependent branches, the count is the same for every input of a given size, which every board run confirmed. Section 8.2 separates core time from the delay through the complete correction loop, and Section 8.5 measures the added cleanup time. Connecting these stages to a quantum controller remains future work (Section 9.3).

Where tested, repeated-inference RTL simulations measure the interval between successive calculations separately from single-frame latency. Physical transactions also repeat, but include serial loading and readback. Neither test

demonstrates a continuing stream from a quantum processor. A core that finishes one frame in C cycles does not necessarily accept the next frame after C cycles.

Resource counts come from Vivado after routing. Block memory is reported in units of one 36-Kibit block. Each 18-Kibit block counts as half a unit, so a design using one block of each size uses 1.5 BRAM36 equivalents. A qualified clock has non-negative routed setup and hold slack under the stated constraints. These checks establish whether each signal arrives early enough before the receiving clock edge and remains stable long enough afterwards. Correct finite board observations at an unqualified clock are reported separately. For larger devices without physical tests, the time is a verified cycle count divided by the frequency supported by internal timing analysis.

4.7 Measuring decoding quality

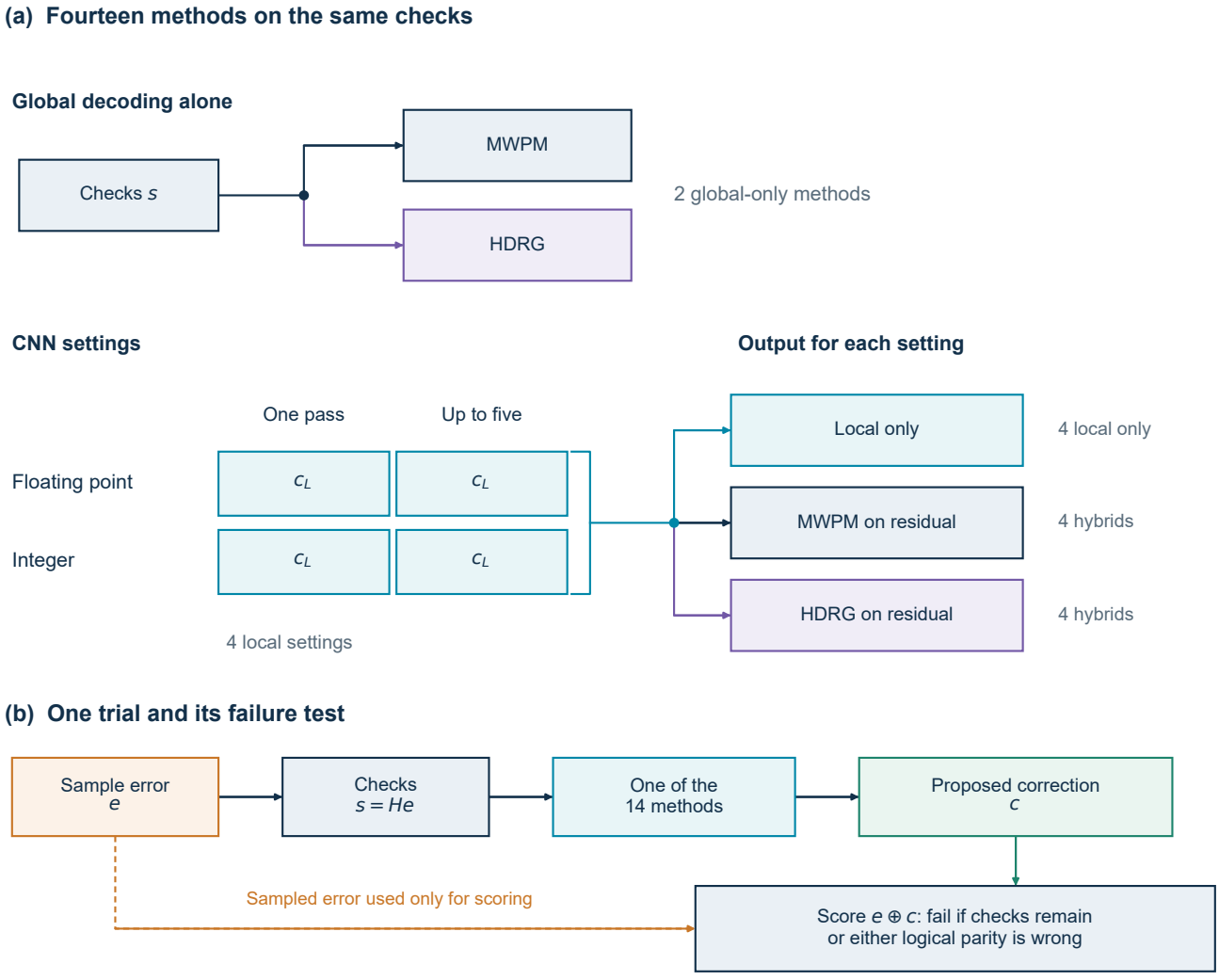


Figure 4.2: Decoder combinations and their evaluation

(a) Two baselines apply MWPM or HDRG directly to the input syndrome s. Four CNN settings combine floating-point or integer arithmetic with one or up to five passes. Each setting can return its local correction alone, or send the residual $s \oplus Hc_L$ to MWPM or HDRG. This gives four local-only and eight hybrid decoders. (b) Only the evaluator sees the injected error. A trial fails if any check remains fired or either logical component is wrong. Chapter 7 also examines correlated matching.

Shared records in paired comparisons. Both methods receive the same sampled errors in simulation or the same measured shots in a device experiment. The global decoder works on the original syndrome, while the hybrid gives it the checks left after the CNN's correction (Figure 4.2). In simulation only the evaluator sees the true error. To measure

what the CNN adds, each hybrid is compared with its own cleanup decoder working alone. The final-readout figures for the IBM quantum processor also use MWPM alone as a common reference. Comparisons that use different sample sizes or scoring rules are identified separately in Appendix C.

In the 2D simulations a trial fails if any check remains fired after correction, or if the error and correction together cause a logical bit flip, a logical phase flip or both. In the 3D simulations and repeated-measurement device experiments, the *window failure rate* is the fraction of trials that give the wrong logical value for the prepared memory after R rounds. In the final-measurement device experiments, the *parity failure rate* is the fraction of shots whose decoded logical parity disagrees with the known preparation. A CNN alone rarely clears every check, so standalone CNN results use the strict rule and are reported separately.

Time boundaries in the 3D experiments. Each simulated window starts with state preparation, runs $R = d$ rounds and ends by measuring every data qubit. Detectors aligned with the prepared basis exist at every time step; the other type has no reference at the first and last step and is masked rather than treated as a measured zero (Figure 4.3).

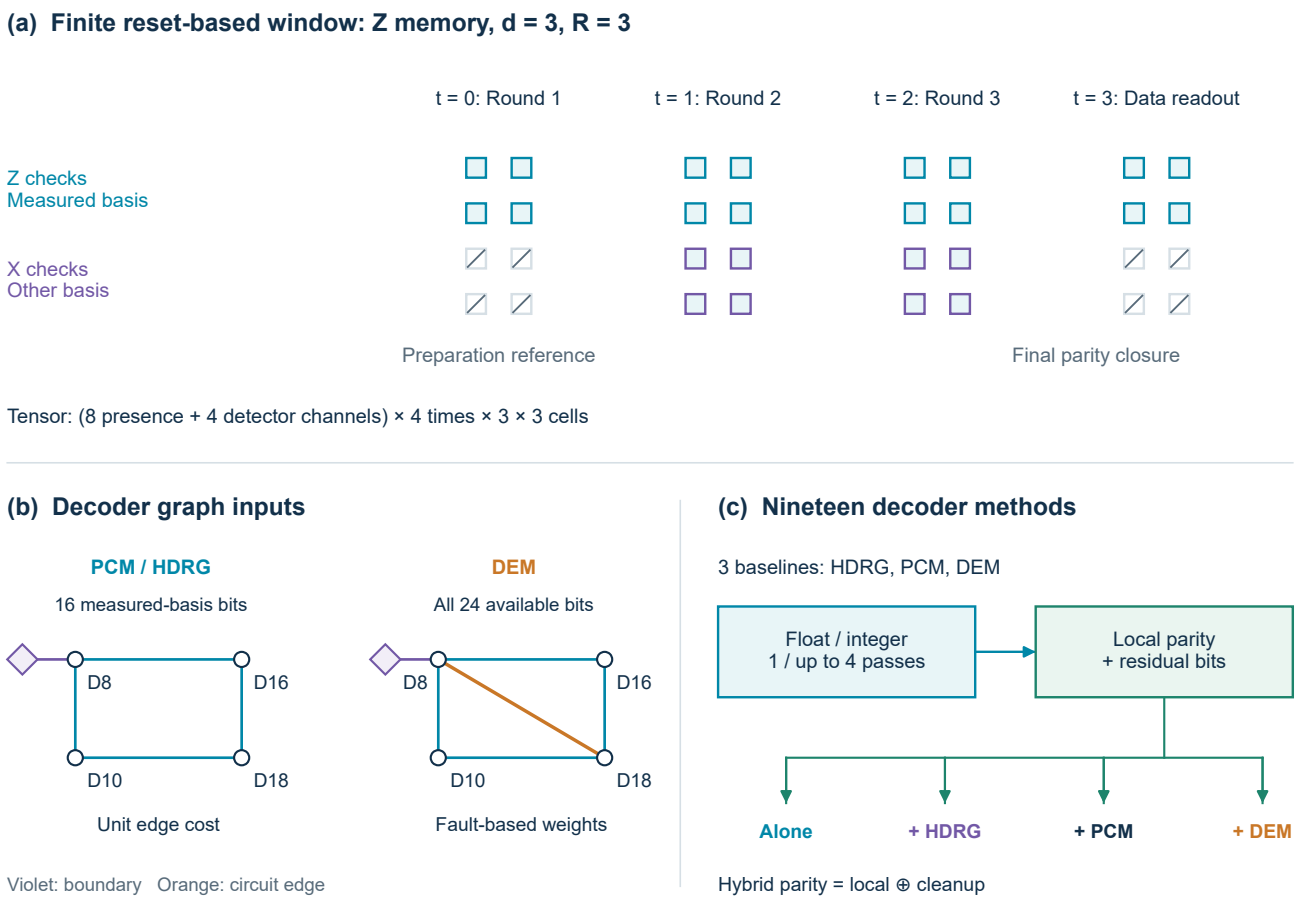


Figure 4.3: Finite detector windows and the 3D decoders

(a) In a rotated $d = R = 3$ memory experiment, detectors in the measured basis exist at every time step. The other type lacks a reference at the first and last step, so those positions are masked. (b) PCM matching and HDRG use measured-basis detectors on a simple graph. DEM matching uses all available detectors (labelled D) with circuit-derived weights. (c) The sweep combines local states with these three cleanup methods. Chapter 7 also compares correlated DEM matching.

The missing entries at the ends of a 3D history have a physical meaning. Consider a memory prepared as a logical zero and read out in the Z basis. At the end, the parity of the measured data bits belonging to each Z check gives its final outcome. This can be compared with earlier ancilla measurements. Those readouts cannot supply a final X parity

for the same shot. Filling the missing slot with a measured zero would indicate to the decoder that a particular check had not fired, although no such observation was made. Only detector positions defined by the circuit contribute to the measured record.

Tuning and evaluation. The number of CNN passes was selected on a separate set of tuning trials. The pass count with the fewest tuning failures was evaluated on fresh trials. Ties were resolved in favour of the smaller count. The sample sizes and random seeds were fixed before each simulation campaign. A trial that helped select the number of passes cannot also provide an independent test of that choice. If two pass counts perform nearly equally well, chance alone may make one appear better on the tuning set. Fixing the pass count before testing measures its performance on inputs that played no part in the choice.

4.8 Failure rates and confidence intervals

A failure rate is the number of failed trials divided by the number of trials. Each rate has a 95% Wilson interval to show sampling uncertainty [55]. For example, zero failures in 4,096 trials gives an upper limit of about 0.094%.

Both decoders see the same trials, so their difference depends only on cases where one succeeds and the other fails. At distance 33 and 13% physical error probability, only CNN+MWPM failed on 2,347 trials, while only MWPM failed on 2,581. The CNN therefore saved 234 failures in 65,536 trials, or 0.357 percentage points. The paired 95% intervals in Chapter 7 use these disagreements [56, 57]. A negative difference favours adding the CNN. An interval crossing or touching zero leaves the direction uncertain. Each interval applies to one comparison under the tested conditions.

5. The 2D decoder on the Arty A7

This chapter follows the 2D decoder from a register-based design that could not fit on the board to an implementation that processes a distance-33 frame in 1.154 ms. The improvements come from how data are stored, how many products are computed together and when each layer can start.

5.1 The workload

The 2D network applies three 11×11 convolutions with $4 \to 9 \to 9 \to 2$ channels to a $(2d-1) \times (2d-1)$ grid (Section 2.5). Across all output channels, one grid position requires 4,356 binary-input products in layer one, 9,801 multiplications in layer two and 2,178 in layer three. That is 16,335 products per position. There are 81 positions at distance 5 and 4,225 at distance 33. The board has 240 DSP multipliers, far fewer than the 9,801 that layer two alone would need to form all its products simultaneously using one DSP per product. The design therefore reuses the arithmetic over several clock cycles.

The first layer needs only to select each weight or zero because its inputs are binary. The second layer accounts for most of the multiplication. Every distance uses the same weights, but each grid size has its own build.

Consider one output of layer two. At each of the 121 locations in its window, the layer reads nine values from layer one. Each of its nine output channels uses a different set of weights on those same values. Reading a neighbour once can therefore serve several calculations. Moving the centre of the window one place to the right changes the addresses to read, while the weights remain in the same storage locations. A larger patch gives this engine more positions to visit. It does not require a new set of weights for each position.

5.2 From registers to addressed memory

Figure 5.1 compares the three architectures built for this network.

Register windows. The first design kept a sliding window of each feature map in registers, making the current position's neighbours available together. The engineering records report bit-exact simulation at distances 5 and 30, although those modules are no longer available for an independent rerun. A hidden pixel holds nine channels of 24 bits (216 bits), and the recorded eleven-row window at $d = 30$ holds 701 pixels. That requires 151,416 flip-flops for one hidden layer, more than the 126,800 on the device. The storage requirement alone rules out this implementation on the Arty.

Shared schedule. The second design stores every feature map in block RAM and fetches each kernel tap by address. All three layers run concurrently, each trailing the previous one by a fixed number of pixel slots so that the neighbourhood it needs has already been written. A single slot counter drives all three layers, one tap per cycle, so one pixel takes 121 cycles. This was the first design to run on the board. It took 22,632 cycles (226.32 µs) at distance 5 and 494,532 cycles (4.945 ms) at distance 30. Every decision matched the integer reference. The common slot length makes the binary first layer wait for the more demanding second layer. More read ports would need more copies of the memories, and at $d = 30$ a single 216-bit-wide hidden map already spans several BRAM blocks.

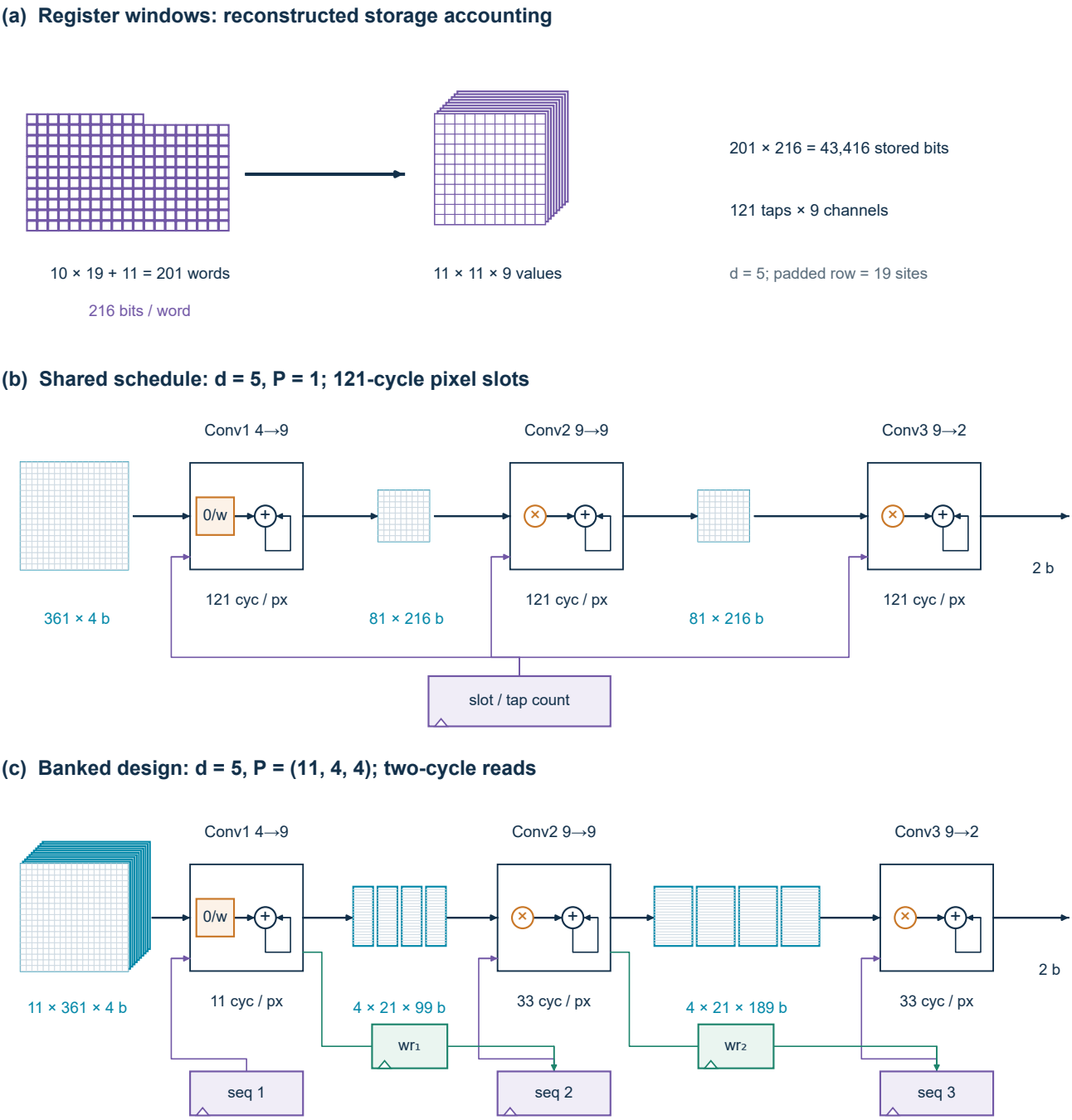


Figure 5.1: Storage and scheduling in the three 2D architectures

(a) The register window at distance 5 holds 201 words of 216 bits. At distance 30 the same design needs 701 words, or 151,416 flip-flops. (b) The shared-schedule design stores feature maps in block RAM and drives all layers from one slot counter, one tap per cycle. (c) The banked design replicates the binary input and splits each hidden map across four banks, so a group of taps can be read in one cycle; each layer has its own sequencer and readiness counter.

Banked, independently scheduled layers. The third design makes two changes. Hidden maps are split across memory *banks* so that several taps can be read in the same cycle without copying the data, and each layer has its own sequencer that advances as soon as its inputs are ready. The rest of this chapter describes this design.

5.3 Banking and the arithmetic pipeline

A *tap* is one of the 121 spatial positions in an 11×11 kernel. A tap lane processes the values and weights for one such position. A configuration is described by its tap lanes per layer, (P_1, P_2, P_3): how many taps each layer processes per

cycle. In the $(11, 4, 4)$ configuration, layer one reads a whole kernel row of eleven taps per cycle and the other layers read four.

A bank is a separate memory with its own address and read port. Splitting consecutive positions across banks lets the lanes read different positions in the same clock cycle. With four banks, position i is stored in bank i mod 4, at address $\lfloor i/4 \rfloor$ within that bank. The example in Figure 5.2 starts at position 14. The group spans positions 14 to 17, so the engine reads banks 2 and 3 at address 3, and banks 0 and 1 at address 4. In bank order the words come from positions 16, 17, 14 and 15. A rotation restores position order so that each value is paired with the correct weight. A word holds all nine channels of one position, 99 bits after layer one and 189 after layer two (Section 4.4).

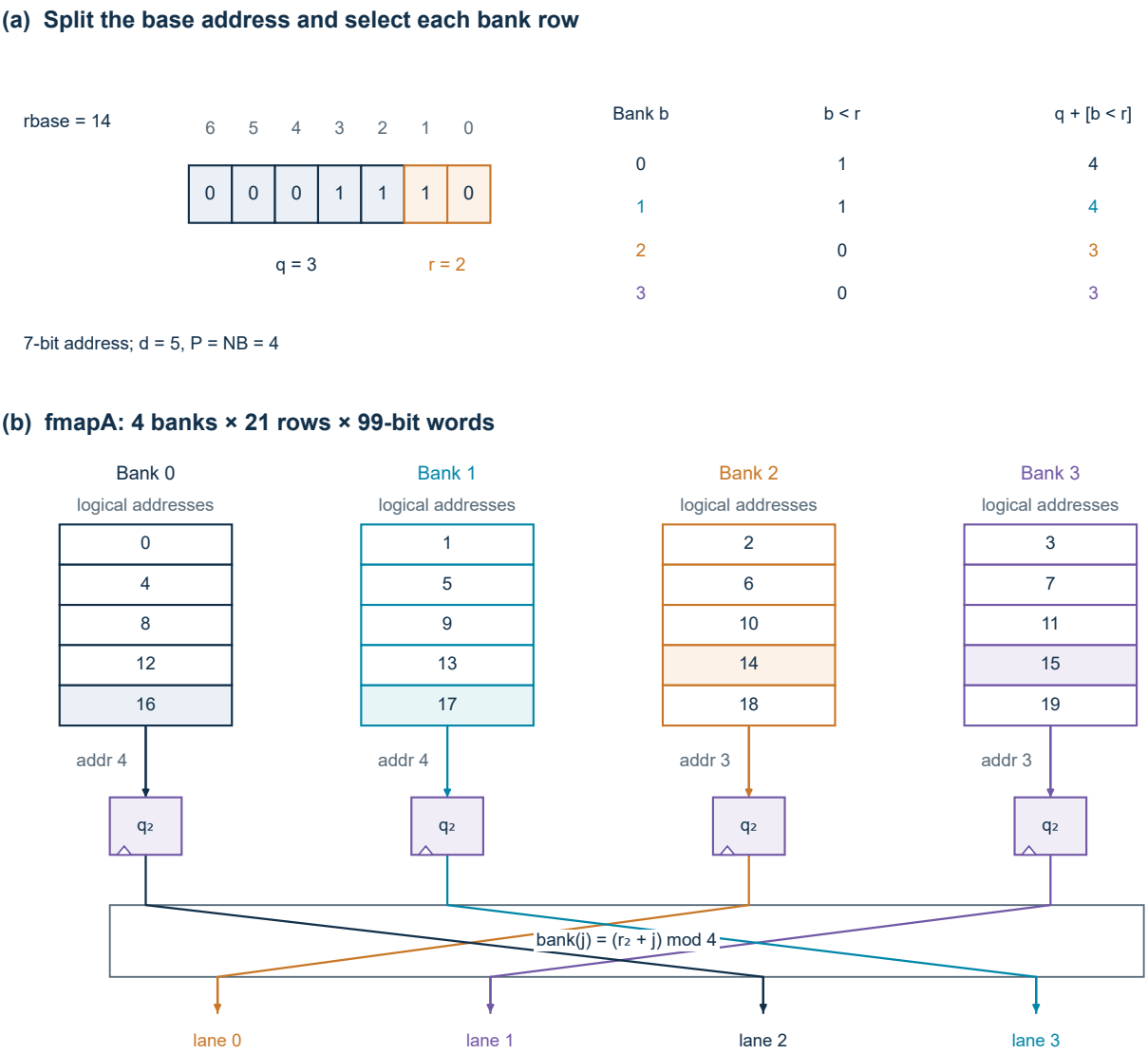


Figure 5.2: Reading consecutive pixels from four memory banks

A distance-5 read from the four-bank layer-one map. (a) Base address 14 gives row $q = 3$ and rotation $r = 2$, so the banks return the values at positions 16, 17, 14 and 15. (b) After the two-cycle memory pipeline, lane j selects bank $(r + j)$ mod 4, restoring the order 14–17. Out-of-range taps are masked downstream.

Each group of taps stays within one kernel row. With four lanes an eleven-tap row takes three groups, the last one partly empty, so a kernel takes $11\lceil 11/4 \rceil = 33$ issue cycles instead of the 31 that perfect packing would allow. The 6.45% overhead provides conflict-free banking and simple boundary handling, and disappears when the lane count divides eleven.

Figure 5.3 follows a group of four taps through layer two. The engine reads the values from memory, forms their products and adds the group sum to an accumulator. Once it has processed the whole kernel, it adds the bias, applies ReLU and writes the hidden value. The four lanes all contribute to the same output position.

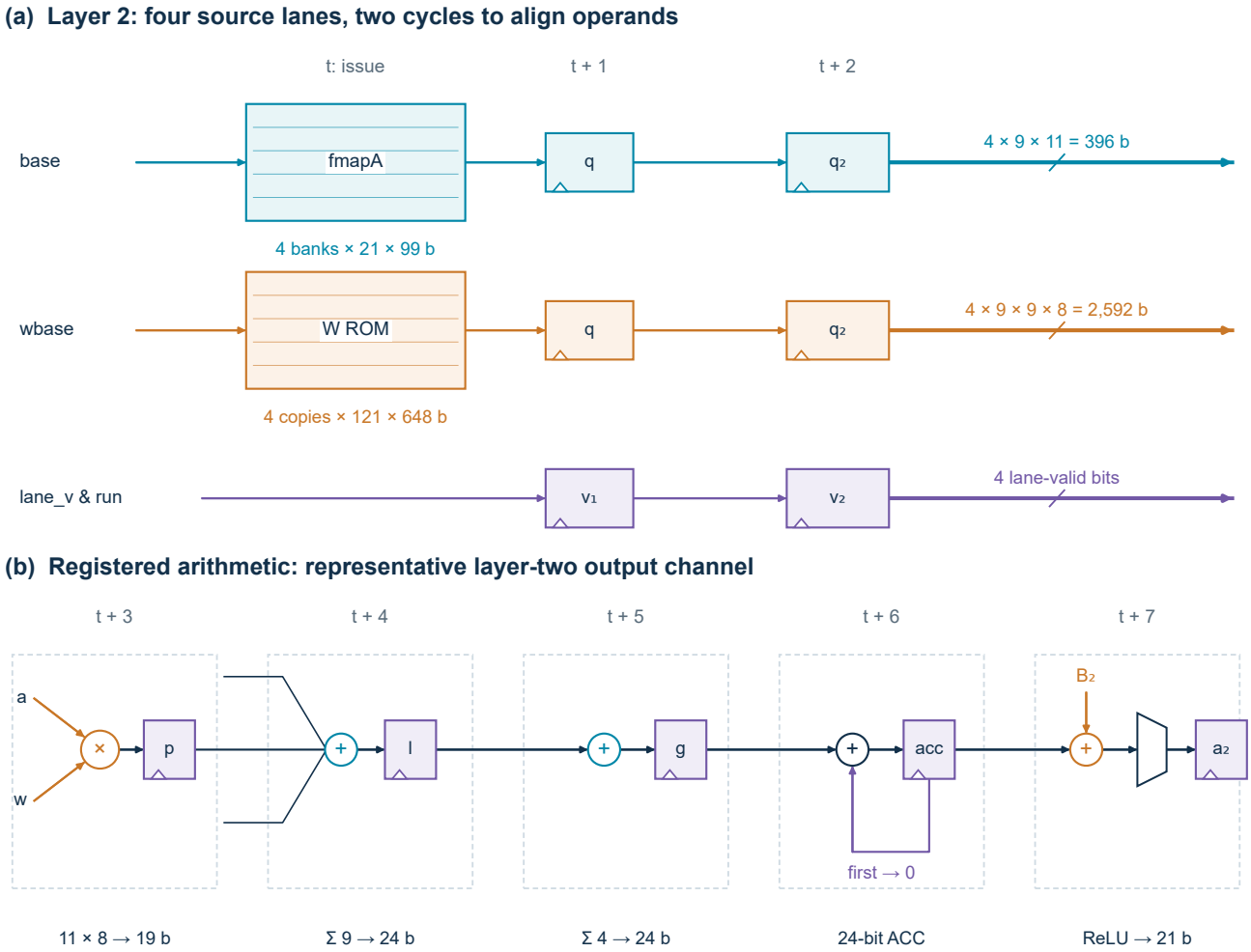


Figure 5.3: Aligned data and control in the layer-two engine

(a) Two registered reads align the activations, weights and valid flags for four taps. (b) For one output channel, the engine forms the products, adds them within each lane and then across lanes, and accumulates over the kernel. The B_2 input supplies that channel's stored bias before ReLU and write-back. (c) The control registers delay the first- and last-group flags and pixel address to match the data.

The clock is common to all the lanes. Adding a lane increases the number of products that the FPGA forms between the same two clock edges. For four taps in layer two, the engine receives four words of nine values each. It multiplies those values by the weights for each of nine output channels and adds the products into nine separate running sums. All nine sums still belong to the same position. On the next clock cycle a different group of neighbours can enter the pipeline while these products are being added.

The control signals must be delayed together with the values. In Figure 5.3, a flag identifies the first group of a new position so that its accumulator is reset. Another identifies the last group, after which the bias and activation must be applied. The position number is delayed with them to select the write address. Without that delay a newly issued

address could receive the previous position's completed value. This is why the figure shows registers for control as well as arithmetic.

The arithmetic is split between DSP blocks and LUT logic. In the $(11, 4, 6)$ configuration, one layer-two lane uses 81 DSP blocks: nine input channels multiplied by weights for nine output channels. All six layer-three lanes use 108 DSP blocks, giving 189 in total; the other layer-two products are built from LUTs. The $(11, 6, 6)$ configuration uses two layer-two DSP lanes and four layer-three lanes, 234 of the 240 DSP blocks. Both compute exactly the same integer sums.

5.4 Scheduling and a cycle model

Each layer has its own sequencer. The layer that writes a map is its *producer*, and the layer that reads that map is its *consumer*. Before a consumer computes position p, it checks that the producer has *committed* (finished writing) every source position it might read, up to $p + 5\,OW + 5$ for a map of width $OW = 2d - 1$. Positions are written in raster order, from left to right along each row and then from top to bottom. The check is therefore a single comparison against the producer's write counter. Near the bottom edge, where future neighbours fall outside the map and are defined as zero, the consumer is released when the producer finishes. The schedule never depends on the error pattern, so every input of a given size takes exactly the same number of cycles.

For a distance-5 map, consider the first output, centred at the top left. Its window extends five rows down and five columns to the right. The last real input it can use is at row five, column five, position 50 when numbering starts at zero. In raster order the first 51 writes therefore cover that entire neighbourhood. The positions above or to the left of the map contribute zero. Waiting for the write counter is necessary because a position already issued for calculation may not appear in memory for several more clock cycles.

The three layers can now work on different positions of the same map. Layer one may already be several rows ahead while layer two uses a neighbourhood it has written and layer three finishes an earlier output.

This makes the run time predictable. With $T_k = 11\lceil 11/P_k \rceil$ issue cycles per kernel in layer k, $NWIN = OW^2$ positions and a lead of $LAG = \min(5\,OW + 6,\ NWIN)$ positions,

$$C = LAG \cdot T_1 + NWIN \cdot T_2 + LAG \cdot T_3 + 21.$$

The first term is the time layer one needs to build the first neighbourhood, the second is the dominant layer-two work, the third allows layer three to finish its remaining positions, and the constant covers the pipeline stages. At $d = 5$ and $(11, 4, 4)$: $OW = 9$, $LAG = 51$, $(T_1, T_2, T_3) = (11, 33, 33)$, so $C = 561 + 2{,}673 + 1{,}683 + 21 = 4{,}938$ cycles. The cap applies at distance 3, where the 5×5 map has only 25 positions. Each layer then starts once the preceding layer has written the whole map. The $(11, 4, 6)$ design takes $25(11 + 33 + 22) + 21 = 1{,}671$ cycles. The expression matches the eighteen-configuration banked-design simulation sweep and every banked board count in Table 5.1. The expression was checked for these lane ratios. The shared design follows a different schedule.

Figure 5.4 separates these four schedule terms. In the $(11, 6, 6)$ design, the middle sweep accounts for about half the 3,486 cycles at distance 5, but almost 90% of the 103,894 cycles at distance 33. This explains why independent layers help most at small distance. The lead terms grow linearly with the map width, while the middle term grows with its area. In the shared design the pipeline fills at the slower layer-two rate. Independent scheduling lets layer one perform this work at the rate of its simpler binary arithmetic. The $(11, 4, 4)$ speed-up is therefore larger at distance 5 (4.58×) than at distance 30 (3.86×).

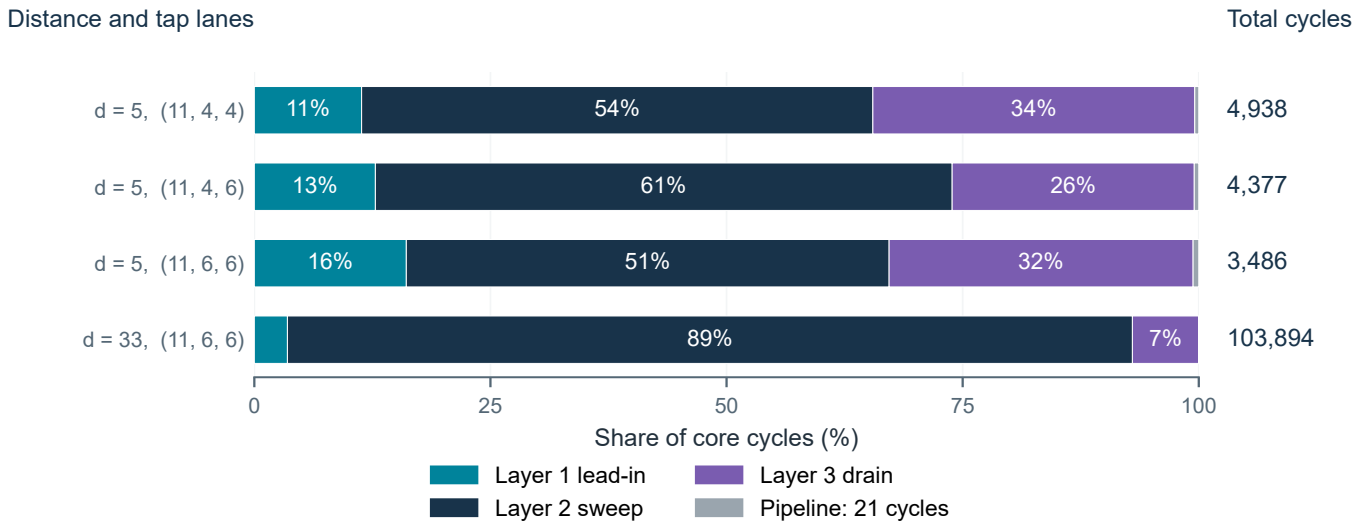


Figure 5.4: Cycle breakdown for the overlapping 2D schedule

Each bar divides the overlapping schedule into the first layer's lead-in, the second layer's sweep, the third layer's drain and 21 pipeline cycles. Bars sum to 100%, with totals at right. The three lane counts give each layer's parallelism. At distance 5 the configurations take 4,938, 4,377 and 3,486 cycles. At distance 33 the $(11, 6, 6)$ build takes 103,894 cycles, about 89% in the middle sweep. The pipeline term is too small to see.

Figure B.2 in Appendix B.2 shows the traced issue intervals of all three layers for the distance-5 shared and banked schedules, so the periods when the layers wait or work together can be read directly from the trace.

5.5 Timing closure

Timing closure means arranging the hardware so that every constrained signal path meets its clock budget. The slowest path between registers limits the clock frequency. The shared-schedule design required four timing corrections. Summing products, accumulating, adding the bias and applying ReLU between the same pair of registers exceeded the clock budget, so a register was added after the adder tree. The added register stores a partial result so that the remaining work can continue in the next clock cycle. Synthesis then rebuilt the tree as an unregistered DSP cascade. The numerical result was still correct, yet the signal passed through too many DSP blocks before reaching a register. Keeping multiplication in DSPs and the summation tree in LUT logic resolved this.

At distance 30, a hidden map spans 21 BRAM blocks. Selecting data from the larger memory introduced another long path. Registering the memory outputs separated the read delay from the depth-selection logic. Replacing a multiplication in the address calculation with constant per-tap offsets shortened a fourth critical path. The kernel offsets are fixed, so they can be computed when the hardware is built instead of being multiplied during every access.

The banked design required a separate control-path correction. Its readiness check combined clipping and additions before a clock enable, and failed the distance-30 timing target by 0.679 ns. An equivalent simpler comparison removed the problem without changing the schedule. These results show that an operation count alone cannot determine the achievable clock frequency, which also depends on register placement, memory layout and routing.

5.6 Results on the board

Table 5.1 lists the configurations tested on the board at timing-qualified clocks. All returned decisions matched the integer reference. The two shared-schedule settings each ran their nominal frame three times. The banked $(11, 4, 4)$ design ran three different frames five times each at distance 5, and one frame three times at distance 30. Each of the seven $(11, 4, 6)$ and $(11, 6, 6)$ configurations passed 1,000 transactions drawn from 67 distinct inputs.

Table 5.1: Board-measured 2D configurations on the Arty A7-100T, with routed resources and worst setup (WNS) and hold (WHS) slack. BRAM is in BRAM36 equivalents. Weight storage uses BRAM in some builds and LUT logic in others, so cross-design memory counts are not a distance-only comparison.

Design	d	MHz	Cycles	Time	LUT / FF	BRAM	DSP	WNS / WHS (ns)
Shared	5	100	22,632	226.32 µs	7,031 / 2,881	18	99	+0.396 / +0.057
Shared	30	100	494,532	4.945 ms	7,058 / 2,955	61	99	+0.351 / +0.033
Banked $(11, 4, 4)$	5	100	4,938	49.38 µs	32,921 / 9,984	94	234	+0.018 / +0.025
Banked $(11, 4, 4)$	30	100	128,138	1.281 ms	42,905 / 16,876	45.5	234	+0.139 / +0.014
Banked $(11, 4, 6)$	5	100	4,377	43.77 µs	40,338 / 20,780	35.5	189	+0.188 / +0.022
Banked $(11, 4, 6)$	9	100	12,561	125.61 µs	40,461 / 20,820	36	189	+0.088 / +0.022
Banked $(11, 4, 6)$	17	100	41,601	416.01 µs	40,471 / 20,869	36	189	+0.125 / +0.023
Banked $(11, 4, 6)$	25	100	87,537	875.37 µs	40,409 / 20,873	42	189	+0.216 / +0.011
Banked $(11, 4, 6)$	33	100	150,369	1.504 ms	40,498 / 20,941	77.5	189	+0.116 / +0.023
Banked $(11, 6, 6)$	5	90	3,486	38.73 µs	44,825 / 16,607	128	234	+0.090 / +0.079
Banked $(11, 6, 6)$	33	90	103,894	1.154 ms	56,389 / 25,704	79.5	234	+0.112 / +0.023

Figure 5.5 compares the distance-5 builds. Banking and independent scheduling reduced the time from 226.32 µs to 49.38 µs. Six tap lanes in the final layer instead of four shortened each kernel row from three groups to two, saving another 11.4% (4,377 cycles). Six layer-two lanes reduced the cycle count further, but the design no longer met timing at 100 MHz: the $(11, 6, 6)$ build failed by 0.515 ns and also failed at 95 MHz. At 90 MHz, generated by the FPGA's clock manager, it met timing and ran in 38.73 µs, 5.84 times faster than the first board design. At distance 33 it took 23% less time than $(11, 4, 6)$, even at the lower clock.

The board comparison also quantifies the cost of a slower clock. At distance 5 the design with a four-lane middle layer takes 4,377 cycles at 100 MHz, or 10 ns per cycle. The six-lane version saves 891 cycles, but each of its remaining 3,486 cycles takes about 11.1 ns. The six-lane version therefore saves about 5 µs per frame on the board. The cycle count alone would overstate the saving because the wider design needs a longer clock period.

Figure 5.6 shows the $(11, 4, 6)$ distance sweep: simulation at 21 distances between 3 and 49, with board measurements at five of them. It also compares lane choices at distance 5. The simulated sweep checked every raw sum, activation and decision. The five-setting board sweep checked 16.17 million decision bits over 5,000 transactions. Independent Icarus Verilog tests of $(11, 4, 6)$ at distances 5, 9 and 17, and $(11, 6, 6)$ at distance 33, also matched the integer reference on separate inputs. Their cycle counts were 4,377, 12,561, 41,601 and 103,894 respectively.

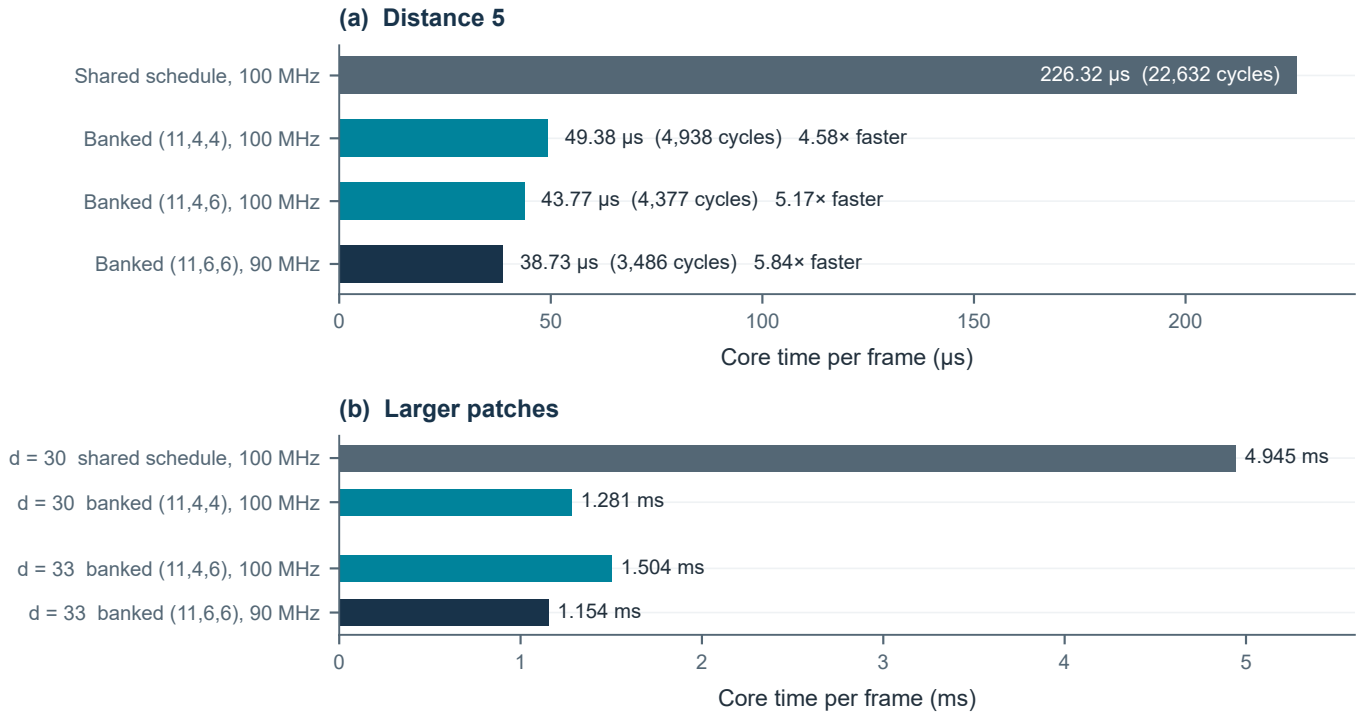


Figure 5.5: Measured 2D inference time on the Arty A7

Board-measured time from starting a loaded frame to its last prediction. (a) At distance 5, banking, independent scheduling and wider layers reduce the time from 226.32 to 38.73 µs. (b) At distances 30 and 33, compare designs within each distance. The lane counts refer to layers one, two and three. Input transfer and global cleanup are timed separately.

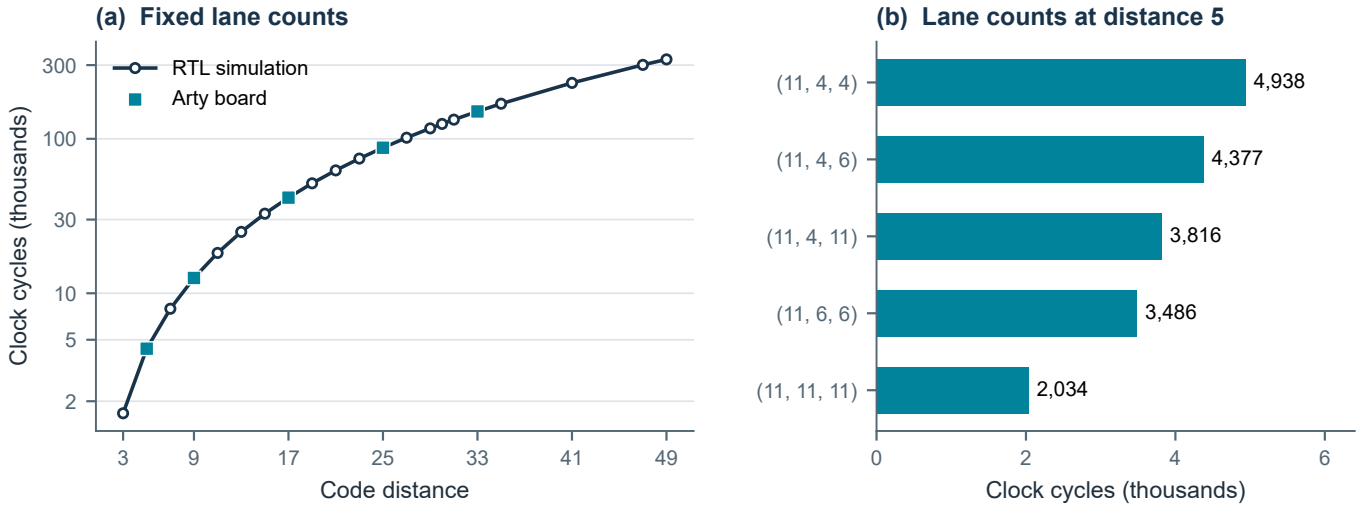


Figure 5.6: 2D core cycles by code distance and lane count

(a) RTL cycle counts for the $(11, 4, 6)$ design at 21 distances from 3 to 49 (logarithmic axis); squares mark the five distances also run on the board, each with 1,000 correct transactions. (b) Distance-5 cycle counts for other lane allocations. Physical fit and clock timing are assessed separately.

Figure 5.7 shows what changes when the grid grows and the lane counts stay fixed. From distance 5 to 33 the $(11, 4, 6)$ design uses the same 189 DSP blocks and about 40,400 LUTs. Its feature maps need more memory, with block RAM rising from 35.5 to 77.5 BRAM36 equivalents. The extra positions are processed by the same multipliers.

The distance-5 grid has nine positions along each side and the distance-33 grid has 65. The number of outputs to visit grows by a factor of about 52. Across those same two board builds, the $(11, 4, 6)$ core time grows by a factor of about 34. The difference comes from the start and end of the schedule. Those parts occupy a larger share of a short run. As the map grows, most clock cycles are spent processing its interior at the middle layer's rate.

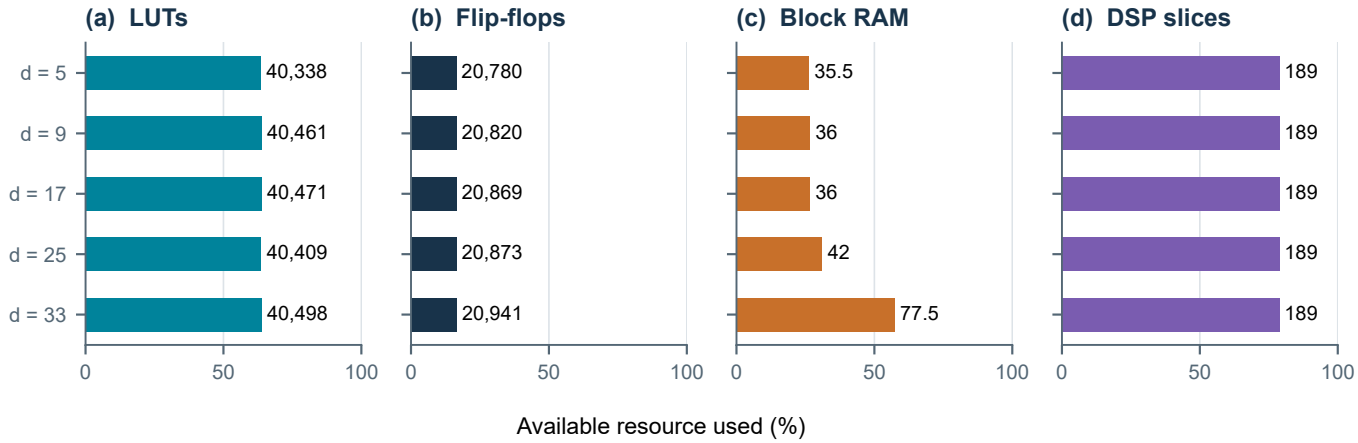


Figure 5.7: Resource use of the banked 2D design by code distance

The five routed $(11, 4, 6)$ builds on the XC7A100T. Bars show the fraction of each resource used. The labels give the corresponding counts, with block RAM counted in 36,864-bit blocks. Because arithmetic is reused across positions, LUT and DSP use change little with distance, while block memory grows with the feature maps. All five meet timing at 100 MHz.

5.7 Throughput and limits on further speed-up

In simulation, with an input already in memory, the $(11, 4, 6)$ core completed 1,000 successive distance-5 calculations at an interval of 4,378 cycles, one cycle more than a single calculation. Reloading each input in the testbench raised the interval to 4,740 cycles, and over the 115,200-baud UART each transaction takes milliseconds. The core finishes each frame before loading the next. Loading the next frame while the current one is being decoded would reduce the gap between results.

At distance 5, simulation gave 3,816 cycles for $(11, 4, 11)$ and 2,034 for $(11, 11, 11)$. Neither configuration both fitted on the Arty and met its target clock. The distance-33 $(11, 6, 6)$ implementation already uses 97.5% of the DSP blocks and almost every logic slice. Appendix B.3 gives the implementation results for the wider designs.

Additional lanes reduce the cycle count only in steps. An eleven-tap row needs two groups with either six or eight lanes, so both take 22 issue cycles to process the eleven rows of a kernel. Eight lanes use more hardware but leave more lanes idle in the final group, without reducing the group count. Eleven lanes reduce it to one group per row. This explains the flat parts of the lane sweep: more arithmetic helps only when it reduces the number of cycles on the layer that limits the schedule.

The 2D results show how memory access and layer scheduling affect speed on a fixed device. The 3D network introduces a different constraint: storing almost a million weights on the same board.

6. The 3D decoder on the Arty A7

The 3D network uses measurement history to help distinguish data errors from measurement errors. It has about sixty times as many weights as the 2D network. At eight bits each, the weights alone exceed the Arty's block memory. This chapter first reduces the storage needed for the two largest layers and then compares ways to run the network faster.

6.1 Workload and memory requirements

The network has three hidden $3 \times 3 \times 3$ convolutions ($12 \to 128 \to 128 \to 128$ channels), a skip connection that appends the twelve input channels to the 128 hidden ones, and a final convolution from those 140 channels to twelve outputs (Section 2.5). That is 971,568 weights and 396 biases, and each weight is used once per voxel. A voxel is one spatial cell at one time slice (Section 6.2). A distance-5, five-round input has 96 voxels, so one inference performs about 93 million multiply-accumulates, nearly all of them in the two middle layers, which hold 442,368 weights each. At distance 33, with 33 rounds of measurements, the same engine would have 11,016 voxels to process.

At eight bits per weight, the coefficients alone need 7,772,544 bits. The board has 4,976,640 bits of block memory. The network would use 156% of that capacity before storing a single measurement or intermediate result.

Changing the code distance would not remove this first constraint. Even for a single output voxel, the middle layers need their full sets of coefficients. Once those coefficients fit, later voxels can reuse them. The intermediate values are different. At each position, layer two needs the values layer one produced for that particular input. Increasing the number of positions therefore increases the memory needed for these maps, even though the coefficients stay the same. The distance-5 build tests whether both kinds of storage fit. The larger builds show which grows as the patch and measurement history grow.

6.2 The input volume

The 3D network receives a regular grid of *unit cells* (Figure 6.1). The rotated surface code repeats with a period of two lattice sites, so a cell at position (t, r, c) can hold up to four data qubits, two X checks and two Z checks. Eight of the twelve input channels mark which sites exist. Near a patch boundary some are absent, and the masks distinguish these from existing checks with no detected change. The other four channels hold the check detection events at that time slice. The twelve outputs propose an X and a Z correction for each data site, and a measurement-error correction for each check. These are local predictions for the cleanup decoder to resolve, rather than direct observations of which physical fault occurred.

For the odd distances tested here, the rotated patch needs $\frac{d+3}{2}$ cells along each side, including partly filled boundary cells. The $R + 1$ time slices cover the R rounds of checks and the final data-qubit measurement. Setting $R = d$ gives $(d + 1)\left(\frac{d+3}{2}\right)^2$ voxels. The convolution engines use this regular tensor layout, while the front end maps the physical code geometry into it. Section 8.1 returns to that interface.

For example, a distance-5 history occupies a four-by-four array of cells on each of six time slices. The same spatial cell has a different entry on every slice. A change detected in round three can therefore appear at the same row and column as a change in round four without overwriting it. A convolution at that position can compare neighbouring

cells in both space and time. The extra time coordinate explains why holding five rounds of distance-5 data provides different information from applying the 2D network to five independent snapshots.

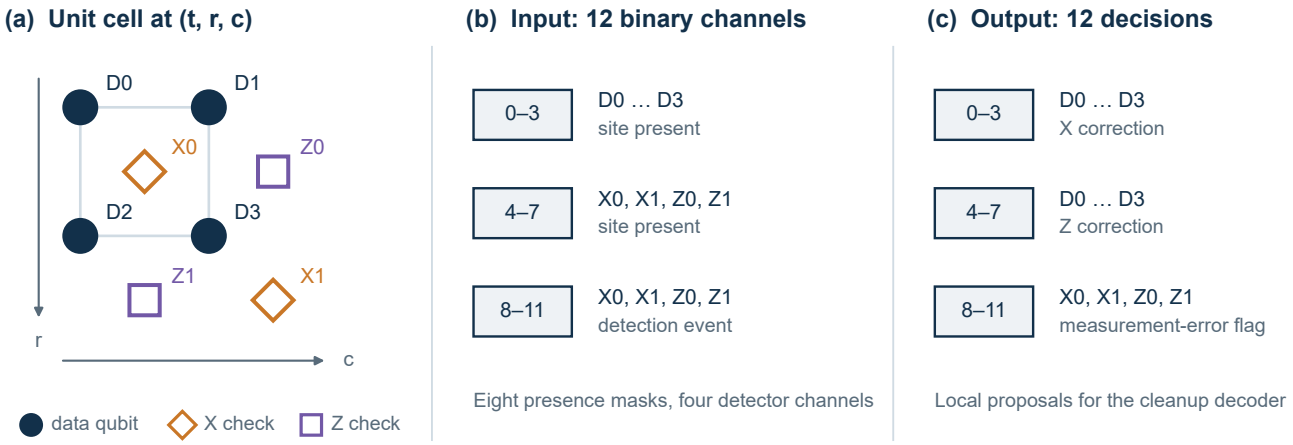


Figure 6.1: Unit cell and channels of the 3D network

(a) The repeating unit cell at position (t, r, c) holds up to four data qubits D0–D3 and four checks X0, X1, Z0 and Z1. (b) The twelve input channels: eight presence masks marking which sites exist, and four detector channels. (c) The twelve outputs: proposed X and Z corrections on the four data sites, and measurement-error flags on the four checks.

6.3 Mixed precision

The first and last layers keep eight-bit weights, and the two middle layers use four-bit weights. With the sixteen-bit skip coefficients, the weights then occupy 4,264,704 bits, 85.7% of the nominal block memory, before biases, scale factors and the packing losses of real memory blocks.

Tests on the same distance-5 data compared each lower-precision network with the floating-point network to select the bit widths. Figure 6.2 shows their *correction overlap*, also known as intersection-over-union. It is the number of correction bits set by both networks divided by the number set by either. Overall bit agreement is misleading here because almost every output bit is zero. The correction overlap was 96.67% when only layer two was reduced to four bits. It fell to 34.75% when only the small output layer was reduced. A uniform four-bit network agreed on 99.58% of bits yet only 46.50% of predicted corrections. The final mixed-precision integer pipeline agreed on 99.9861% of bits and 96.11% of corrections. These numbers measure agreement with the floating-point network. Section 7.2 measures decoding quality directly.

To see what the agreement measures, consider 1,000 candidate correction bits. Suppose the floating-point network marks sites 4 and 12, while a lower-precision network marks sites 4 and 18. The predictions agree on 998 bits. In the set of sites either network would correct, however, only site 4 is common to both. The correction overlap is therefore one third. Counting the many zeros would conceal the change in the proposed correction. This was particularly important when choosing the precision of the last layer, which directly decides which sites to mark.

The scheduling tests use the same mixed-precision network. Both designs therefore compute the same integer answer. A difference in cycle count can be traced to how they execute the network.

Appendix B.4 records an additional check of the classifier trained with simplified correction labels. The board results and complete-decoder comparisons here use the original, unsimplified model.

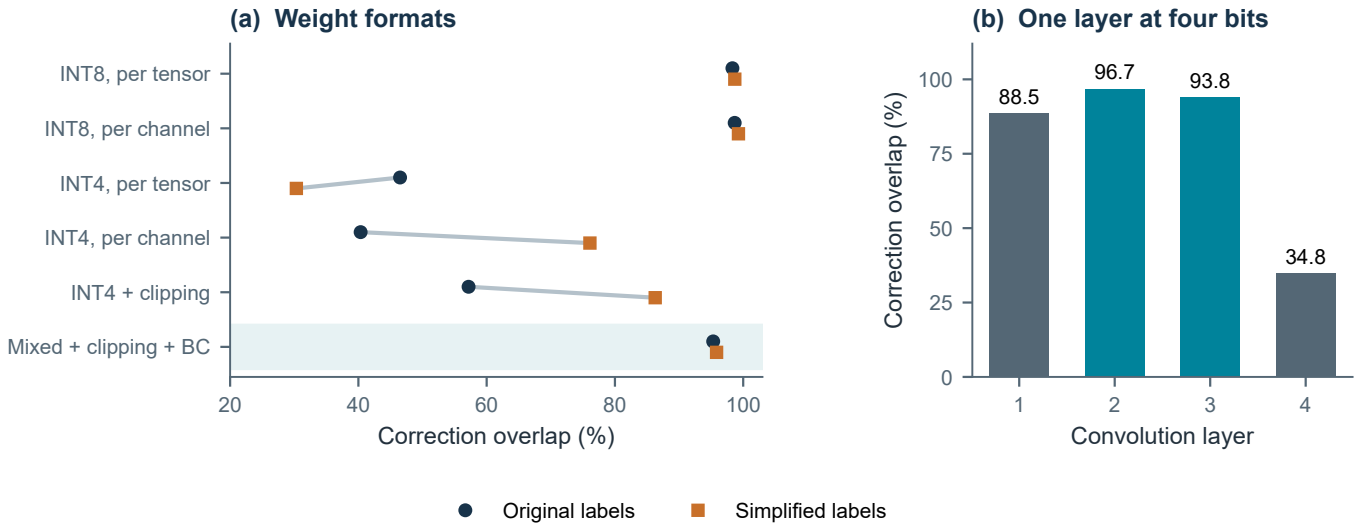


Figure 6.2: Effect of weight precision on 3D predictions

Agreement with the floating-point network on fixed distance-5 inputs. Correction overlap is the number of corrections both networks propose divided by the number proposed by either, expressed as a percentage. (a) Original- and simplified-label networks under uniform and mixed precision. Scales apply per tensor or per channel. BC denotes bias correction. (b) Changing one layer at a time to four-bit weights has the largest effect in the output layer. Teal bars mark the two middle layers, which use four-bit weights on the board; the first and last layers keep eight-bit weights. The board uses the original-label network.

6.4 Four engines and channel banking

The hardware has one engine per layer, and the four engines run concurrently on different voxels (Figure 6.3). Each engine reuses a small arithmetic *tile* over kernel taps and channel groups. Here a tile is a group of multipliers and accumulators reused within one voxel calculation. The spatial tiles assigned to separate boards in Section 8.6 have a different meaning. A hidden engine consumes eight input channels at a time and accumulates TCO_H output channels together, where TCO_H is 16 or 32 in the Arty designs. Weights are stored in the order they are needed, simplifying their read addresses. In the fixed-slot baseline, LUTs multiply four-bit weights by eight-bit activations, and each hidden requantiser uses one DSP block. The requantiser converts one finished channel per cycle while the next group is accumulated. The readiness-based builds use a different arithmetic mapping, reported below.

For example, a middle layer takes 128 inputs and produces 128 outputs at each voxel. With sixteen outputs active, an engine holds sixteen running sums. At one kernel tap it reads eight inputs at a time, so sixteen reads cover all 128 inputs. After 27 taps, the sixteen sums are complete. Eight groups of sixteen give all 128 outputs. With 32 outputs active, only four groups are needed. The stored weights keep the same precision in both designs.

Each hidden feature map holds 128 eight-bit channels for every voxel. It is split into sixteen *channel banks*, each storing an eight-channel, 64-bit slice per voxel, so a consumer reads the bank matching its current input-channel group. One 64-bit read supplies all eight channels needed for that step. This is a different kind of banking from Section 5.3: there, banks separated neighbouring positions; here, the address selects the voxel and the bank selects the channel slice. The input is kept in two copies, one for the first layer and one for the skip path, because the last layer needs the original input long after the first layer has finished reading it.

The routed sixteen-channel design shows how memory width and depth affect the number of blocks required (Figure 6.4). Each middle-layer weight ROM supplies a 512-bit word. Vivado mapped it to 57 RAMB36 blocks: 56

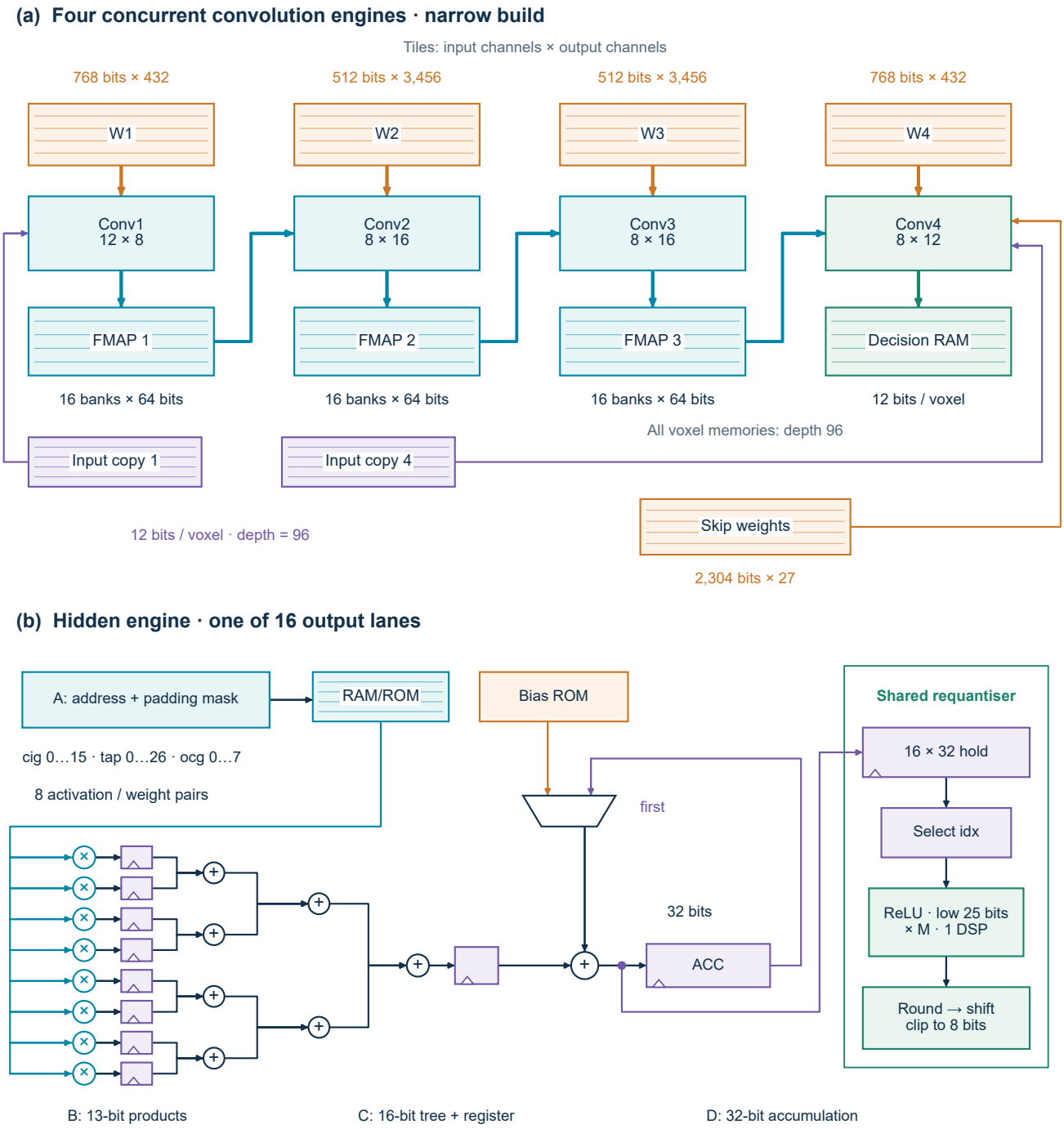


Figure 6.3: The four concurrent 3D engines

(a) The narrow build shown here computes sixteen hidden output channels in parallel. At distance 5, each 96-voxel feature map is held in sixteen 64-bit channel banks. Two copies of the binary input serve the first layer and the skip connection, whose sixteen-bit weights have their own ROM. (b) One of sixteen hidden lanes multiplies eight unsigned 8-bit activations by four-bit weights and accumulates in 32 bits; a shared requantiser applies ReLU, scaling, rounding and clipping.

contribute nine bits each and the last contributes eight. Each block provides 4,096 addresses, while the stored coefficients use 3,456. The payload alone equals 48 blocks of storage, but the required width and depth use 57 in this mapping.

The two middle-layer ROMs therefore use 114 RAMB36 blocks. Other mapped memories use nineteen RAMB36 blocks and one RAMB18 block, giving 133.5 of the Arty's 135 BRAM36 equivalents after routing. The hidden maps and the decisions use distributed LUT memory.

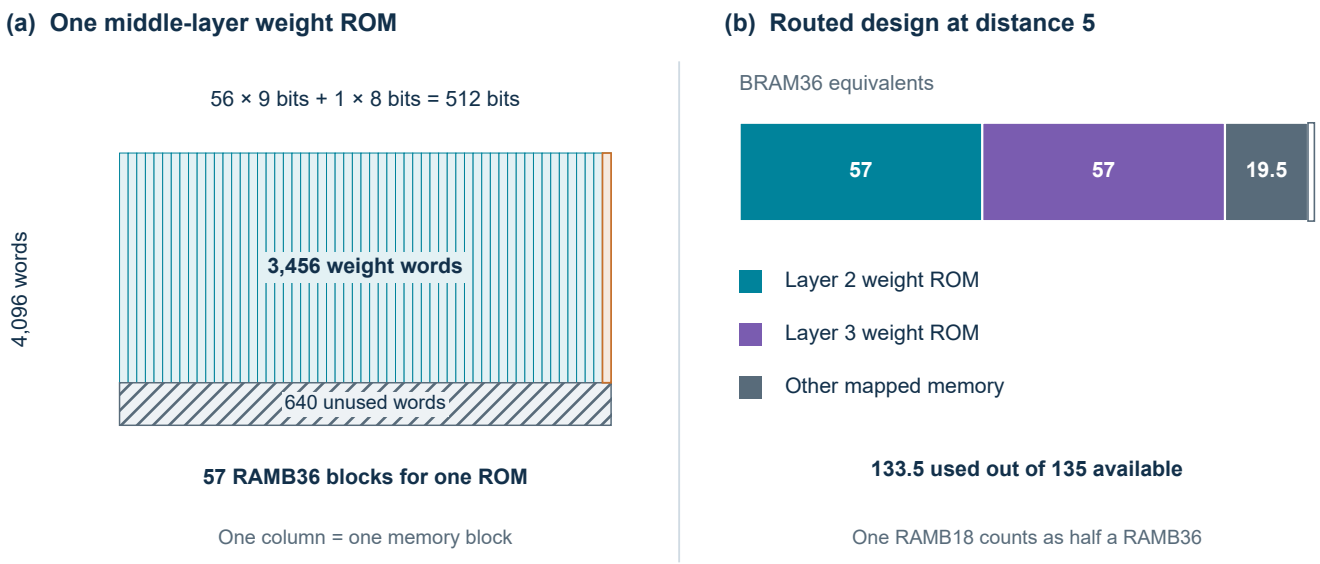


Figure 6.4: Block-memory mapping in the 3D design

(a) Vivado maps each middle-layer weight ROM to 57 parallel RAMB36 blocks: 56 supply nine bits each and the last supplies eight. Each block has space for 4,096 words, of which 3,456 hold weights. The hatched region is unused depth. (b) The routed distance-5, 16-channel design uses 57 blocks in each middle layer and another 19.5 block equivalents elsewhere. The total is 133.5 of the 135 available. The final partly used block in (a) leaves one of its nine bit positions unused.

Most unused capacity in a middle-layer ROM lies at the end of its address range. A different feature map cannot use that space under this layout. Its reader needs its own address at the same time as the engine needs a particular row of coefficients. The total bit count therefore overstates the storage that this design can use. This is why the routed block count, alongside the coefficient count, determines whether the maps also fit.

6.5 Fixed-slot and readiness-based scheduling

The first 3D design advanced all four engines in lockstep, one voxel per *slot*. The slowest engine sets the slot length. For the hidden engines it is

$$SLOT = 27 \times 16 \times \frac{128}{TCO_H} = 3{,}456 \text{ cycles for } TCO_H = 16,$$

counting kernel taps, input-channel groups and output-channel groups. A consumer must also trail its producer by enough voxels for the neighbourhood to be complete. With voxels stored row by row within each time slice and $n_r = n_c = (d + 3)/2$, that lag is $LAG = n_r n_c + n_c + 2$ slots, or 22 at distance 5. With $NVOX$ voxels, the complete schedule is

$$C = (NVOX + 3\,LAG - 1) \cdot SLOT + 463,$$

where 463 cycles finish the output stage. This expression applies when the hidden engines set the slot length, as they do for $TCO_H \leq 64$. At distance 5 this gives $(96 + 66 - 1) \times 3{,}456 + 463 = 556{,}879$ cycles, exactly what the board reported. Doubling TCO_H halves the slot only up to a limit: at $TCO_H = 128$ the hidden engines would need 432 cycles per voxel, below the unchanged output engine's 459, which would then determine the slot length.

The fixed slot leaves the faster engines idle. The first and last engines need only 432 and 459 cycles per voxel, but each waits a full slot, and filling the pipeline costs 3 LAG slots at the slow rate. In the *readiness-based* design each engine starts its next voxel as soon as its own previous voxel is complete *and* every source voxel it needs has been committed, meaning all sixteen channel banks for that voxel have been written. Out-of-range neighbours are zero. The arithmetic, the weights and the decision rule are unchanged. At distance 5 the sixteen-channel design fell from 556,879 to 427,475 cycles (23% fewer), and the 32-channel design from 278,671 to 223,603 cycles.

RTL tests of both designs at ten distances from 5 to 33 compared every raw accumulator and stored activation with the integer reference, checked output order and data availability, and tested reset recovery and repeated inputs. These tests covered 138 complete windows and more than 123 million checked values. A second simulator that tracks unknown logic values independently checked selected configurations, and four deliberately corrupted tests were correctly rejected.

In the 2D core, a single write stored all nine hidden channels for a position. Here a producer writes the groups for a voxel over several clock cycles. If it announced that voxel as soon as the first group arrived, a consumer could read valid low-numbered channels and old values from a later group. A voxel is ready when its final group has been written. This gives the next engine a simple rule for when it can read a whole neighbourhood.

6.6 Results on the board

Table 6.1 lists the 3D builds that ran on the board. The fixed-slot design met timing at 100 MHz only after higher-effort implementation and changes to its controller and summation trees. Both readiness-based designs assign arithmetic to LUTs and 99 DSP blocks to reduce logic use. The 32-channel design runs at 95 MHz from the FPGA's clock manager. Each readiness-based build passed 1,000 complete UART transactions over 67 distinct inputs, including reference volumes, all-zero and all-one volumes, a boundary impulse and random patterns. Every returned decision matched the integer reference. The recorded replies were also checked independently.

Table 6.1: Board-measured 3D configurations at distance 5 with five rounds (96 voxels). "ch." is TCO_H, the number of hidden output channels computed together.

Design	MHz	Cycles	Core time	LUT / FF	DSP	BRAM36	WNS / WHS (ns)
Fixed slot, 16 ch.	100	556,879	5.569 ms	47,934 / 17,023	3	133.5	+0.091 / +0.008
Readiness, 16 ch.	100	427,475	4.275 ms	41,609 / 14,907	99	133.5	+0.003 / +0.026
Readiness, 32 ch.	95	223,603	2.354 ms	52,738 / 21,209	99	133.5	+0.003 / +0.014

The fastest design processes a volume 2.37 times as fast as the fixed-slot baseline. Both readiness-based builds meet their clock constraints with a worst setup margin of only 0.003 ns, or three picoseconds. This is the time left between the latest predicted signal arrival and the setup deadline, so these builds have very little timing margin. Their longest paths pass through the feature-memory reads. About 70% of the delay on those paths comes from routing. Figure 6.5(a) shows the measured times at the frequency at which each design met timing.

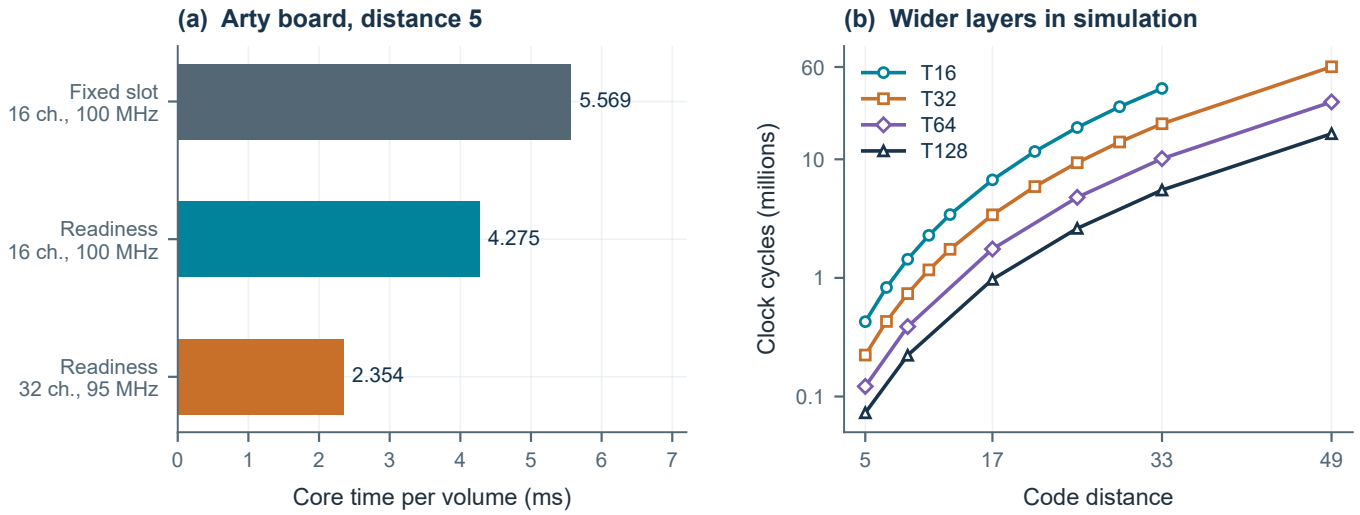


Figure 6.5: Measured 3D core time and simulated cycle counts

(a) Distance-5 core times measured on the Arty at the clock frequency each design met after routing. (b) Simulated cycle counts with one check round per unit of code distance ($R = d$), for tiles of 16, 32, 64 and 128 hidden output channels (T16 to T128). The vertical axis is logarithmic. At distance 49, going from 64 to 128 channels gains only 1.85 times, because the unchanged output engine becomes the bottleneck.

Two other builds ran on the board but did not meet their timing targets (Appendix B.3). The fixed-slot $TCO_H = 32$ design needed a different arithmetic mapping to route, and then failed timing at 100 MHz by 0.324 ns. It returned correct decisions in 60 board runs. A distance-7 fixed-slot build returned 2,400 correct decisions in each of twenty runs and completed in 1,019,983 cycles, as predicted. Its larger volume needed a deeper distributed memory, doubling the LUTs used for storage from 8,352 to 16,704. The build failed timing by 2.137 ns. The successful test inputs do not establish reliable operation at either unqualified clock.

The board tests also exposed an error outside the convolution engines. The UART wrapper (Appendix B.1) receives a complete volume, starts the core and returns the counter and the packed decisions. After one calculation the completion flag could still be set when the wrapper requested the next. The wrapper sometimes read that old flag and returned the previous result. Repeating the same input would give the expected decisions in either case. Alternating two different volumes over 300 transactions made the error visible. The wrapper waits until the core acknowledges the new start before accepting a completion. Removing that wait reproduces the fault in simulation. A complete transaction over the 115,200-baud link takes tens of milliseconds.

6.7 Larger volumes and larger devices

RTL simulation tested how the readiness-based design scales with wider tiles (Figure 6.5(b)). With $TCO_H = 64$, the hidden engines compute 64 output channels together. Tests of this width, together with the matched $TCO_H = 32$ distance-49 comparison, agreed with the integer reference in more than 408 million raw-sum and activation checks. Appendix D.1 gives the coverage of these and the 128-channel tests. At distance 49, a volume of 50 time planes of 26×26 cells, the run takes 30,438,881 cycles, against 60,250,273 for $TCO_H = 32$: almost exactly half. At $TCO_H = 128$ it takes 16,426,873 cycles, only 1.85 times faster again, because consecutive output voxels are now exactly 459 cycles

apart: the unchanged output engine has become the bottleneck, as the slot formula predicts. At that size the three hidden maps alone hold about 104 million bits.

At distance 49 there are enough voxels to see the limit directly in the output trace. The hidden engines can prepare the next result while the last engine is busy, but each completed result must still pass through that last engine. Doubling hidden arithmetic changes how long the last engine waits for work. Once the waiting time reaches zero, the interval between results is set by its own 459 cycles of work. Shortening that interval would require a change to the output calculation as well.

These wider designs exceed the Arty's resources. Two complete cores passed through placement and routing on larger devices (Appendix B.5). The distance-49, $TCO_H = 64$ core routed on a Versal Premium VP1802, storing its maps in UltraRAM, and met internal timing at 83.3 MHz, so the verified cycle count corresponds to about 365 ms for a preloaded volume. The distance-5, $TCO_H = 64$ core routed on a Virtex UltraScale+ VU13P at 100 MHz: about 1.22 ms. Neither met its original 250 MHz target. These times combine simulated cycle counts with routed clock constraints. Neither core was run on a physical board.

Each of these 3D cores loads a complete volume before decoding. A live decoder would instead receive one round at a time. It would need to keep each slice until every layer, including the skip path, has finished reading it. It would also need a rule for when a correction is final. Even with faster arithmetic, this centred network needs four future rounds to decide about the current one. Section 8.2 explains the resulting delay, and Section 8.11 discusses the storage needed for a continuing stream.

7. Decoding quality with the neural pre-decoder

The board tests in Chapters 5 and 6 establish that the FPGA reproduces the integer predictions. This chapter evaluates whether those predictions improve recovery of the protected qubit. Each input was processed by two decoding pipelines. One used the global decoder alone. The other ran the CNN first and sent the remaining checks to the same global decoder. The tests use three sets of inputs: simulated errors with perfect checks, simulated circuits with noisy measurements, and records from the IBM quantum processor `ibm_miami`.

A negative difference between a hybrid and cleanup alone means fewer failures with the CNN. Differences are in percentage points (pp). A difference is described as unresolved when its paired 95% confidence interval includes zero. The sample does not establish which method performs better. Sections 4.7 and 4.8 describe the paired trials and intervals. The comparison data give the baseline and trial counts.

7.1 2D simulations with perfect checks

In this setting each data qubit independently has an X, Y or Z error with probability $p/3$ each, giving a total error probability p. The checks are measured perfectly, and a trial fails if any check is left fired or either logical component is flipped. A Y error flips both components, so the two error types are correlated. The main one-pass comparison uses $p = 0.13$, high enough that every decoder still fails often enough at large distances to be compared precisely.

The error probability applies to each data qubit separately. A distance-5 patch has 41 data qubits, so at 13% noise a typical frame contains several faults. At distance 33 there are 2,113 data qubits, and the expected number of faults is about 275. The larger patch can still have a lower logical failure rate. The outcome depends on whether the combined error and correction cross the patch in a way that changes the stored information. A large number of short errors can be corrected successfully, while a smaller number in an arrangement that the decoder misinterprets can lead to failure.

7.1.1 One CNN pass

The comparison uses 65,536 fresh error patterns at each of eight distances from 9 to 49. Each hybrid runs the CNN once. The integer version runs the same arithmetic that was timed on the FPGA. Comparing its decisions with the floating-point version shows the effect of conversion. The upper panels of Figure 7.1 give the failure rates. The lower panels show how much adding the CNN changed each rate on the same trials.

HDRG shows a substantial improvement. At distance 33, HDRG alone fails on 27.0% of trials. With one floating-point CNN pass it fails on 12.9%, a reduction of 14.1 pp [13.7, 14.5]. The integer CNN, which is what the FPGA computes, gives 13.3%. The gain grows with distance, reaching 17.6 pp at distance 49.

MWPM shows a smaller improvement. At distance 33, MWPM alone fails on 5.49% of trials and with the floating-point CNN on 5.13%: a reduction of 0.36 pp [0.15, 0.57]. The floating-point hybrid is better at all eight distances, and each of those eight intervals excludes zero (Figure 7.1(d)). The integer hybrid gives 5.26%, a reduction of 0.23 pp [0.02, 0.44], and is resolved as better at six of the eight distances. At distances 17 and 21 its estimated difference also favours the CNN, but the interval includes zero.

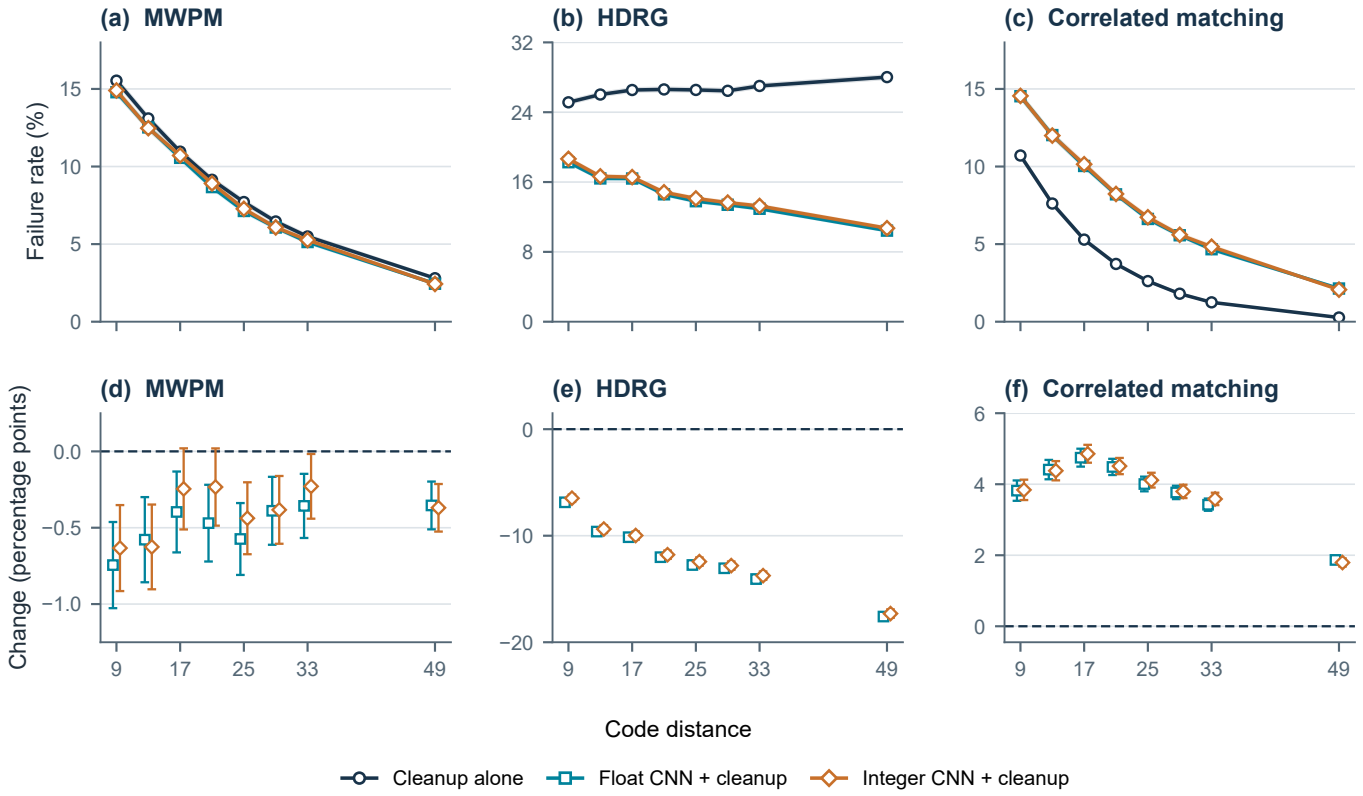


Figure 7.1: Effect of one CNN pass on 2D decoding failures

A trial fails if any check remains fired or either logical component is wrong. (a–c) Failure rates for each cleanup decoder alone and after one floating-point or integer CNN pass. Shading gives 95% Wilson intervals. (d–f) Change in failure rate on the same trials, in percentage points, with paired 95% intervals. Points below zero favour adding the CNN. All methods use the same 65,536 error patterns at each distance and $p = 0.13$. Vertical scales differ between panels.

The rates can also be expressed as expected counts of failed trials. For every 1,000 distance-33 patterns drawn under these conditions, the observed rates correspond to about 270 failures with HDRG alone and 129 after a floating-point pass. The same 1,000 patterns would give about 55 failures with MWPM alone and 51 after that pass. These are averages calculated from the full sample. For a particular pattern, adding the CNN can change a success into a failure as well as a failure into a success. The plotted rate reflects both changes.

The outcome also depends on the noise level. At distance 33, one floating-point CNN pass improved MWPM at the sampled probabilities $p \geq 0.13$. At $p = 0.10$ it increased failures by 0.10 pp [0.04, 0.18]. Below that, too few trials failed to determine which decoder performed better (Section 7.1.2).

Correlated matching performs better without this CNN. Correlated matching, which accounts for a Y error affecting both kinds of check, fails on only 1.25% of trials at distance 33. Running the CNN first raises this to 4.67%, an increase of 3.4 pp [3.3, 3.6], and the same occurs at every distance. At distance 49 correlated matching fails on 0.27% of trials, compared with 2.80% for MWPM. Using these correlations directly gives a larger benefit than adding the CNN to MWPM.

The decoder that gains most from the CNN need not have the lowest final failure rate. At distance 33, the CNN reduces HDRG failures by about half, yet correlated matching alone still has the lowest failure rate. The HDRG improvement agrees with the direction reported by Gicev et al. [10, Sec. 2.4]. Their threshold comparison used up to five passes

and varied the noise. This one-pass distance sweep holds the noise fixed. Section 8.5 examines the separate question of how much time the combined hardware needs.

A fresh-sample check used separately written geometry, sampling, CNN and matching code, retaining the same HDRG implementation. Across 27 rates, the largest difference from the main study was 2.1 estimated standard errors. The check reproduced the broad ranking, although its smaller distance-33 sample left the MWPM benefit unresolved. Appendix D.2 gives the counts and paired comparisons.

7.1.2 Multiple CNN passes

Gicev et al. evaluate up to five CNN passes before cleanup [10]. The noise-and-pass sweep covers thirteen distances from 5 to 49 and eight error probabilities from 0.03 to 0.16. Its 104 conditions contain 671,744 shared trials: 16,384 per condition when $p = 0.13$ or $d = 33$, and 4,096 otherwise. The one-pass comparison above uses a separate set of 524,288 trials. Repeating a decoder on the same inputs does not add independent samples.

The left column of Figure 7.2 shows the effect of adding one CNN pass, or allowing up to five, before cleanup at distance 33. Each of the eight noise levels has 16,384 shared trials. The comparison uses the floating-point CNN so that the effect of repeating the network can be examined separately from integer conversion. A trial stops receiving further passes as soon as all its checks have cleared. At $p = 0.13$, adding up to five passes almost doubled the failures relative to MWPM alone (921 to 1,821 of 16,384 trials). With HDRG the same change reduced failures (4,389 to 2,288). Allowing up to five passes increases MWPM failures at every sampled noise level from 0.08 upwards, by as much as 8.1 pp at $p = 0.145$. For HDRG, one pass and up to five passes both improve recovery over that range. Below it, failures are too rare to compare.

A second pass uses the same weights on the checks left by the first. The second pass receives no new quantum measurements. It may clear a pattern left by the first pass or alter a correction that would already have succeeded. Each additional pass also requires the updated checks to be formed and the CNN to run again. The FPGA times in Chapters 5 and 6 cover one pass.

At each distance, 4,096 tuning trials determined which of one to five passes caused the fewest failures. Ties were resolved in favour of the smaller count. The chosen count was then tested on 8,192 fresh trials. This gave one pass for MWPM and correlated matching at distances 9, 17, 33 and 49. For HDRG it gave three passes at distance 9 and one elsewhere.

Allowing zero passes in the same selection makes the baseline choice explicit. The tuning procedure selects correlated matching alone at every distance, while the HDRG and integer-CNN-plus-MWPM choices stay unchanged. For floating-point CNN-plus-MWPM it selects one pass at distances 9, 17 and 33, but no CNN at distance 49. There, the tuning counts were close (126 failures without the CNN against 135 with one pass), and the evaluation sample favoured one pass instead (186 against 213). The selection uses tuning data only. This reversal illustrates why a small advantage in the tuning sample is not, by itself, a reliable guide.

One possible reason for the multi-pass penalty is that matching weights designed for the original errors describe the CNN's residual poorly. To test this, residual error rates at each site were estimated from the tuning trials and converted

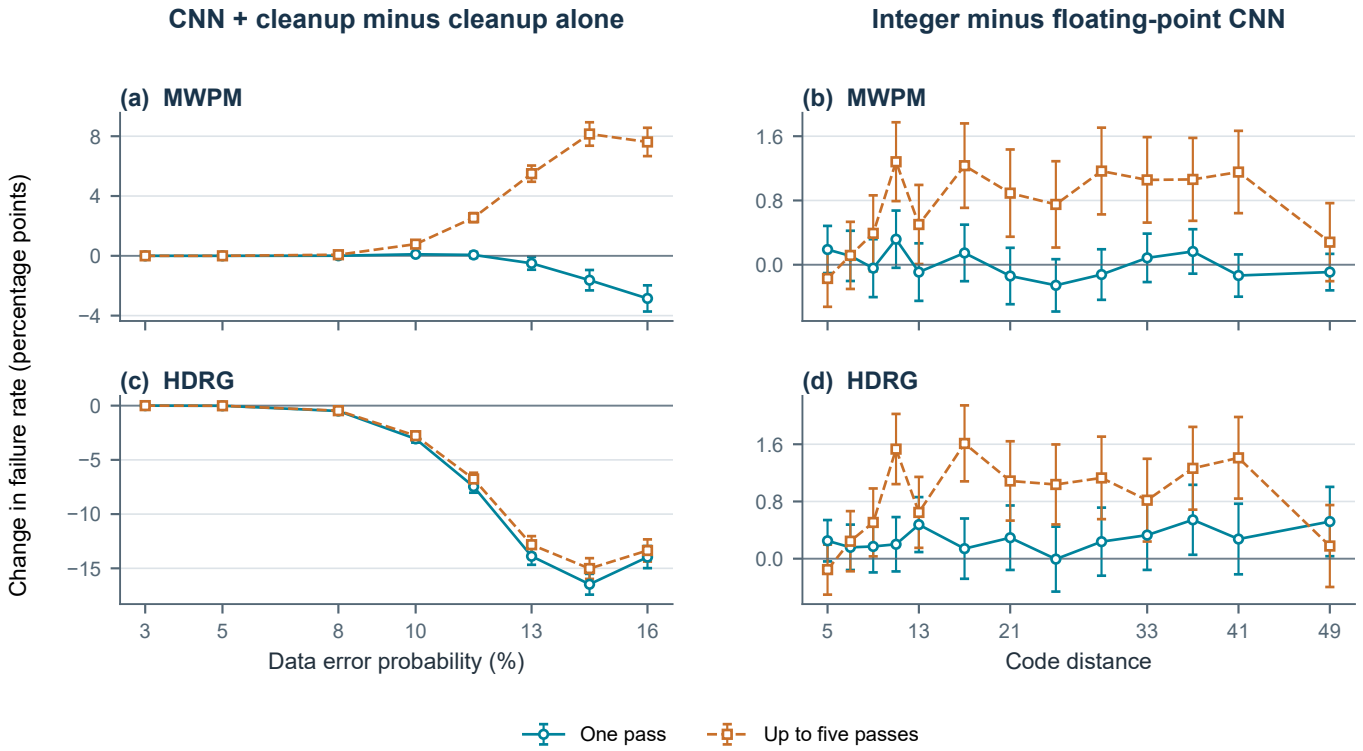


Figure 7.2: Effects of CNN pass count and precision on failure rate

Paired changes in failure rate, in percentage points, for MWPM (top) and HDRG (bottom). In the left column, the floating-point hybrid is compared with cleanup alone at distance 33 as the data error probability varies. A negative value means the CNN helps. In the right column, integer and floating-point hybrids are compared at $p = 0.13$ across code distances. A positive value means the integer hybrid failed more often. Each comparison uses 16,384 shared trials. Bars show paired 95% confidence intervals.

to matching weights. The effect was small and inconsistent: at distance 33 the five-pass floating-point hybrid rose from 933 to 940 failures out of 8,192. This simple change to the weights did not remove the penalty. A decoder that also models correlations in the residual may behave differently.

7.1.3 Integer versus floating-point inference

The FPGA reproduces the *integer* network, so its decoding quality is the relevant measure for the hardware implementation. The right column of Figure 7.2 compares integer and floating-point hybrids at $p = 0.13$. Each pair was tested on the same 16,384 trials at each of the thirteen distances. With one pass, the difference was unresolved at every distance with MWPM and at ten of thirteen with HDRG, with a median cost of 0.25 pp for HDRG. With up to five passes, the integer network was worse at nine distances with MWPM and ten with HDRG, by about 1 pp. These results support the selected integer format for the tested one-pass conditions, while leaving small differences uncertain.

A small change in an output near the correction threshold can change a decision bit even when the FPGA computes the intended integer result exactly. The global decoder may still recover the logical qubit. For that reason the precision comparison scores the final logical outcome after cleanup.

If a decision differs on the first pass, the integer and floating-point networks see different checks on the second. That is one reason the five-pass comparison can give a different result from the one-pass test.

7.2 3D simulations with circuit noise

The 3D study adds noisy measurements. Each simulation prepares a rotated surface-code memory, runs $R = d$ rounds of the eight-step reset-based check circuit and measures all the data qubits [58]. The depolarising circuit-noise model has fault probabilities $3p'/4$ for single-qubit gates and $15p'/16$ for two-qubit gates. Resets and measurements flip with probability $p'/2$. The main comparison uses $p' = 0.005$, within the network's training range.

A *Z-memory* circuit prepares a logical zero and measures the data in the Z basis at the end. The detector record helps interpret that final readout. A logical bit flip changes the expected answer. A phase flip of a logical zero leaves its Z readout unchanged. An *X-memory* circuit prepares a logical plus state and measures the data in the X basis. A logical phase flip then changes the expected answer. In both experiments, a failure means the decoder gives the wrong logical bit at the end of the window.

In the 2D experiment the evaluator has the complete injected spatial error and can test both logical components of a correction on the same frame. Here the final readout supplies one logical observable. The simulator knows which faults it injected, but measuring the two memories requires different preparations and different final measurements. Adding their failures together would count outcomes of separate experiments. It would not show how often either component failed on one shared shot. The two columns of Figure 7.3 preserve that distinction. As $R = d$, the failure rate at each distance also covers a different number of rounds.

The three cleanup approaches use different descriptions of the observed detection events. **HDRG** groups nearby detection events and joins their clusters. **PCM matching**, short for parity-check-matrix matching, pairs detection events on a simple graph of neighbouring positions and rounds. **DEM matching**, short for detector-error-model matching, builds its graph and weights from possible faults in the extraction circuit. It can represent connections and probabilities that the PCM graph omits [11, Sec. 2.3.1]. Correlated DEM matching also uses relationships between errors caused by the same fault.

For example, a single gate fault can cause several checks to fire. The DEM records which combinations each possible fault can produce and how likely it is. Matching must still infer the underlying faults from the measured checks. The simulations use the same circuit to generate faults and to build the DEM. On a device, errors in the estimated noise probabilities affect those weights.

The primary comparison uses 4,096 windows per basis at distances 9, 17, 25 and 33, with one CNN pass (Figure 7.3). The large HDRG improvement contrasts with the much smaller differences between the stronger matching decoders. Table 7.1 compares all four cleanup choices at distance 33, including PCM matching.

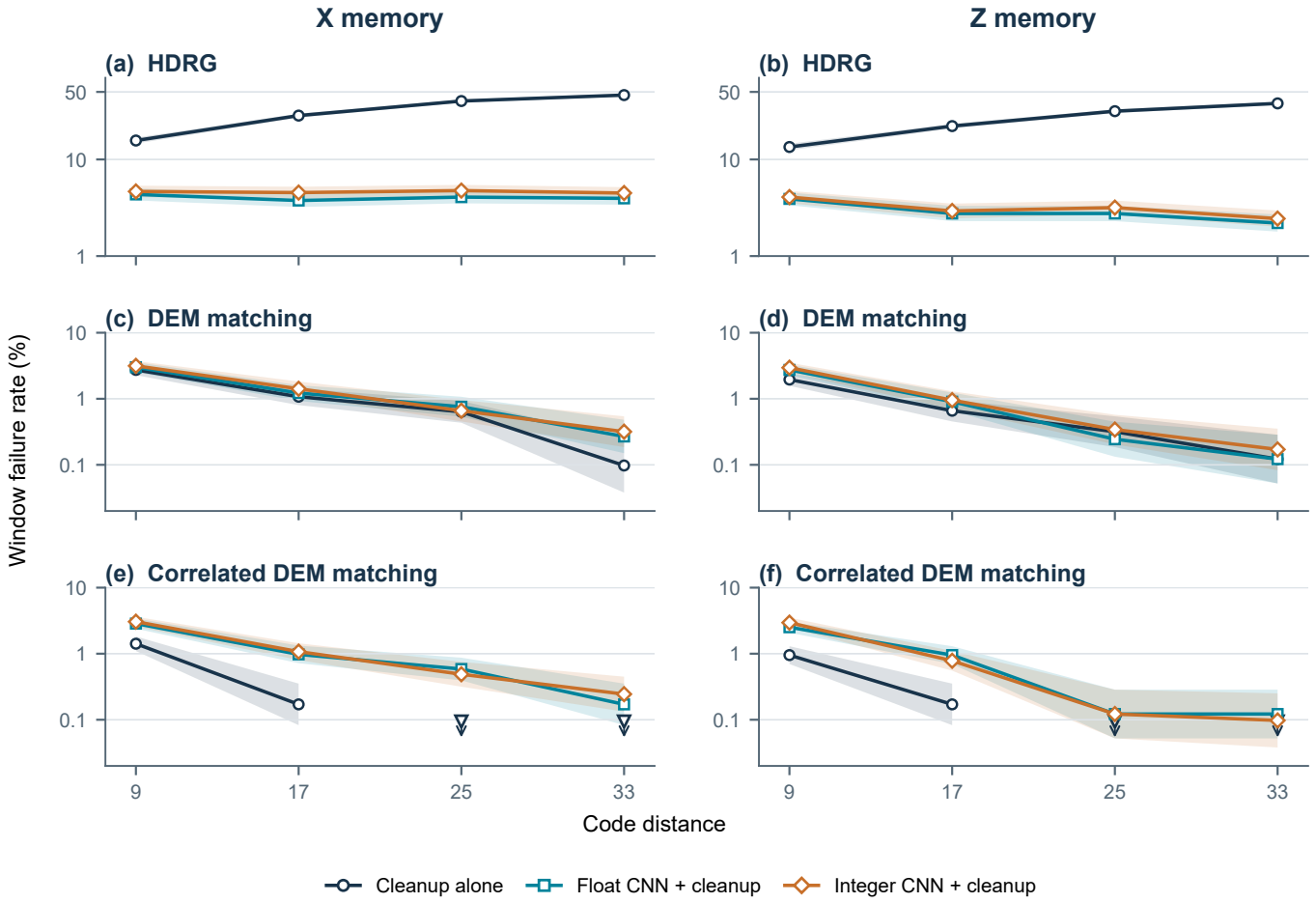


Figure 7.3: Effect of one CNN pass on 3D memory failures

Each column shows failures of the prepared logical value after a complete X- or Z-memory window. The rows compare HDRG, DEM matching and correlated DEM matching, alone and after one floating-point or integer CNN pass. Each comparison uses the same 4,096 windows per distance and basis, with $R = d$ and $p' = 0.005$. Shading gives 95% Wilson intervals. Zero failures cannot appear on the logarithmic scale, so a downward marker shows the upper confidence limit for that point.

Table 7.1: Distance-33 failure counts in the 3D simulation. Each cell gives X-memory followed by Z-memory failures out of 4,096 windows in each basis, with $R = d = 33$ and $p' = 0.005$. The hybrid columns use one CNN pass.

Cleanup decoder	Alone: X / Z	CNN float: X / Z	CNN integer: X / Z
HDRG	1,899 / 1,558	162 / 90	184 / 100
PCM matching	314 / 182	23 / 17	31 / 18
DEM matching	4 / 5	11 / 5	13 / 7
Correlated DEM matching	0 / 0	7 / 5	10 / 4

HDRG and PCM matching have many more failures when used alone, and one CNN pass removes a large part of that gap. With HDRG, for example, the X-memory count falls from 1,899 to 162 with the floating-point CNN and to 184 with the integer CNN. Both numerical formats give these simpler cleanup methods a large benefit.

Whether a CNN helps depends on the cleanup decoder. For a system already using PCM matching, adding the integer CNN reduces the X-memory failures from 314 to 31. If the cleanup method has not yet been chosen, DEM matching alone gives four X-memory failures on the same windows. The additional neural stage can improve an existing decoder substantially while another decoder gives better recovery without it.

Both forms of DEM matching already have low failure counts. Their low initial failure rates leave little scope for improvement, and the CNN introduces additional failures in several cases. For the floating-point correlated X-memory comparison the increase is 0.17 pp [0.08, 0.35]. These few failures suggest a small penalty in this X-memory comparison. Estimating its size across both bases would take more data. The multi-pass comparisons below contain more failures.

Gicev et al. report that the original-label classifier improved recovery with PCM matching and generally offered little benefit with DEM matching [11, Sec. 2.3.2]. Their separately trained iterative model is a different decoder. Repeatedly applying the classifier here does not reproduce that model [11, Sec. 2.3.4].

7.2.1 Interpretation of the distance trends

The distance curves show how failure changes as the code grows. In Figure 7.3, floating-point CNN-plus-HDRG X-memory failure stays near 4% across distances 9 to 33, while HDRG alone rises from 15.7% to 46.4%. The hybrid's advantage grows mainly because it avoids the large increase seen with HDRG alone. Its own failure rate does not steadily decrease over these distances. Each larger code also stores the state for more rounds because $R = d$, so the plot combines a larger patch with a longer noisy experiment. A near-flat curve can occur close to a decoder's threshold, but this finite set of distances at one noise level is insufficient to locate that threshold. The curve shows that increasing distance alone gives little additional benefit to this hybrid in the main comparison.

The 2D curves should be read separately. Their checks are perfect and they describe one spatial frame. Floating-point CNN-plus-HDRG failure still falls from 12.9% at distance 33 to 10.4% at distance 49, while MWPM and correlated matching also continue to improve. These methods therefore show different distance trends under the same 2D noise model. Some 3D DEM comparisons had no failures in 4,096 windows. A larger sample would be needed to estimate such small failure rates.

7.2.2 Multiple passes under circuit noise

The wider noise-and-pass sweep covers eleven distances from 3 to 33, both memory bases and $p' = 0.003$, 0.005 and 0.007, always with $R = d$. These 66 conditions contain 94,208 windows: 2,048 per condition when $p' = 0.005$ or $d = 33$, and 1,024 otherwise. The primary one-pass comparison uses a separate 32,768 windows, and the pass-selection and noise-sensitivity tests use a further 25,600.

At distance 33 and $p' = 0.005$, four floating-point passes were compared with cleanup alone on 2,048 windows per basis. In X memory, the CNN reduced HDRG failures from 917 to 121 and PCM matching failures from 190 to 62. With DEM matching, failures rose from 3 to 39, an increase of 1.76 pp [1.20, 2.46]. In Z memory the DEM count rose from 2 to 12, an increase of 0.49 pp [0.15, 0.94].

Comparing the one-pass and four-pass outcomes on individual X-memory windows shows where the additional failures arise. With DEM matching, 36 windows that succeeded after one pass failed after four. None changed in the other direction. All 36 had fewer remaining detection events after four passes. Across all windows the mean fell from 612.5 events to 120.1. Clearing more checks left a smaller residual, but the final correction was wrong more often. Used

alone, the CNN left checks active in every window at this setting, and its logical parity was correct only about half the time.

Pass selection, with 512 tuning windows and 1,024 evaluation windows per basis, confirmed the pattern. For HDRG at distance 33, tuning chose two floating-point passes in X memory and four in Z memory. On the evaluation windows these gave 47 and 39 failures, worse than one pass (37 and 24) but far better than HDRG alone (458 and 390). A choice that performs best on one finite sample need not remain best on the next. For PCM matching at distance 33, tuning chose one pass, which reduced failures from 66 and 49 to 9 and 2.

7.2.3 Sensitivity to the noise model

Gicev et al. included four idle-noise steps on the data qubits in each round of their circuit [11, Sec. 2.2.2]. Removing them gives a lighter-noise variant. At distance 33, HDRG then fails on 90 X-memory and 30 Z-memory windows out of 2,048, and PCM matching on 4 and 0. DEM matching, correlated DEM matching and every CNN hybrid have no observed failures, so this variant cannot rank the stronger decoders. Changing the noise model alone can therefore change the apparent benefit substantially. Distance trends must be interpreted together with the model that produced them.

7.3 Measurements from the IBM quantum processor

The device experiments test whether the behaviour observed under simulated noise also appears in measurements from a quantum processor. The available qubits and connections limit the code distances that can be tested, but even these smaller patches reveal effects absent from uniform depolarising noise. Comparing preparations, idle times and check rounds helps identify how those effects influence decoding.

The measurement circuits were run on `ibm_miami`, a 120-qubit IBM quantum processor, through the IBM Quantum Hub at the University of Melbourne. All decoding took place offline. In simulation the code distance can change while the noise model stays fixed. On the processor, different patches occupy different qubits, and preparation, gates, idle time and readout can all affect the record. For a fair comparison, the decoders received the same measurements for each setting.

7.3.1 Final-measurement experiments

The first set of experiments prepares a known state on a patch of data qubits, waits for a programmed idle time, and measures every qubit (Figure 7.4). In Z memory the prepared bits are all zero, all one or a fixed random pattern. In X memory each qubit is prepared and measured in the X basis, with or without an XY4 dynamical-decoupling sequence during the wait. Bell experiments pair each data qubit with a neighbour. Comparing the measured bits with the known preparation gives an effective error pattern, from which the surface-code syndrome is computed classically. There are 82 configurations of 8,192 shots each: 58 at distance 5 and 24 spanning distances 5 to 8 (Appendix C.4). These final measurements provide effective error patterns for testing the decoders. Repeated, non-destructive checks are tested separately below.

The data-only patches reached distance 8. The repeated-check circuits also needed space for the qubits that measure the checks, so the largest patch used in those experiments was smaller. Appendix C.6 gives the qubit counts and recorded placements.

XY4 applies a fixed sequence of X, Y, X and Y pulses during the idle period to counter some unwanted phase evolution. It uses no syndrome measurements or feedback, so it complements decoding rather than replacing it. In the Bell experiment, the preparation is reversed before measurement. Errors on either partner, in preparation or inversion gates, or in readout can change the measured outcome. The inferred bit-like and phase-like components therefore do not identify a unique physical fault on the data qubit.

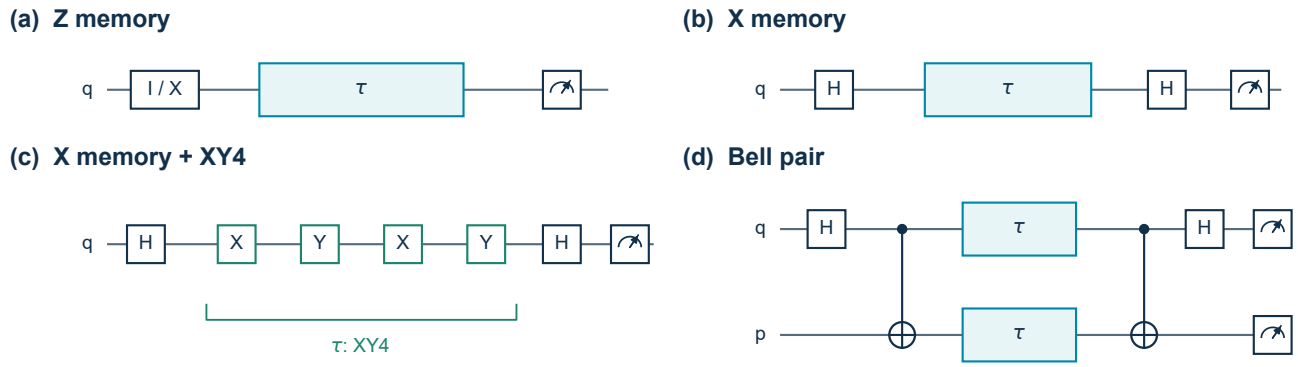


Figure 7.4: Final-measurement experiments on the IBM quantum processor

Circuits for (a) Z memory, (b) X memory, (c) X memory protected by XY4 dynamical decoupling and (d) Bell pairs. Each prepares a known state, waits for a programmed idle time τ and measures every qubit once. Final measured bits are converted into a syndrome in software.

For each Z- or X-memory shot, a raw error is a measured bit that differs from the bit expected from the preparation in that basis. The raw error frequency is the fraction of these disagreements over all measured data qubits and shots, before decoding. It includes preparation and readout errors as well as changes during the wait. These frequencies depend strongly on the preparation (Figure 7.5).

All-zero preparations stay near 2.5% from zero to 160 µs, while all-one preparations rise from 4.2% at zero delay to 17.5% at 40 µs and 50.1% at 240 µs, consistent with relaxation of the qubits towards the ground state. X-memory errors rise from 3.7% to 29.1% within 20 µs of free evolution, but only to 11.3% with XY4. Bell records show that bit-like and phase-like errors are not independent: at zero delay they occur together 3.7 times more often than independence would predict, and the ratio changes with the idle time and with decoupling (Appendix C.3). A single error probability per qubit, as in the 2D simulations, does not describe these records.

The point at zero idle time still includes state preparation and measurement. To measure how errors accumulated in storage, each comparison varied the wait and kept the preparation and measurement basis fixed. Comparing the all-zero and all-one series at the same wait instead probes how strongly the result depends on which state was stored. Combining those two series into a single average would hide the large difference between them.

For decoding, each shot's measured disagreements are assigned to the data-qubit positions of the 2D code. A Z-basis disagreement supplies a bit-flip component, an X-basis disagreement supplies a phase-flip component, and Bell readout supplies both. The corresponding surface-code checks are then calculated in software. Each decoder receives those

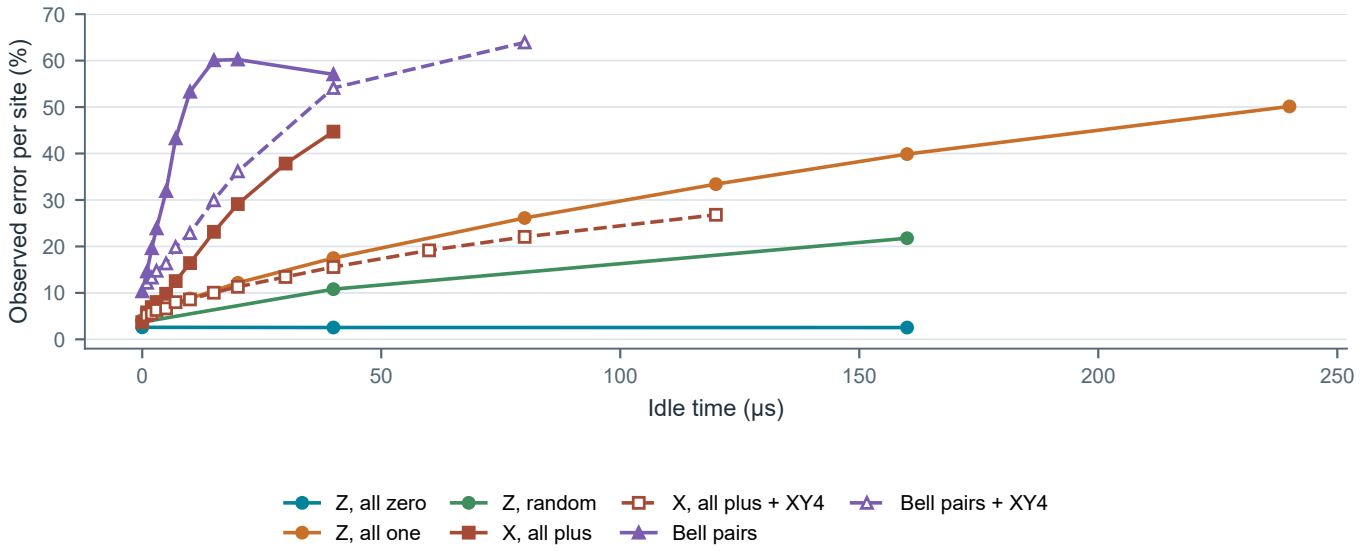


Figure 7.5: Error per site during idle time on ibm_miami

Raw error frequency (observed error per site) against idle time at distance 5, with 8,192 shots per point. All-zero preparations change little, all-one preparations relax towards zero, and X memory decays quickly unless protected by XY4. For Bell pairs the error includes bit-like, phase-like and joint outcomes.

same checks, and the final correction is scored against the same measured pattern using logical parity. No further random errors are sampled. This tests the 2D decoder on device-derived error patterns, rather than demonstrating storage of an encoded quantum state.

Figure 7.6 compares these outcomes with the integer CNN applied once before MWPM or HDRG. For the all-zero preparation at zero delay, MWPM alone fails on 39 of 8,192 shots and the hybrid on 36, a difference of -0.04 pp $[-0.18, +0.11]$. HDRG fails on 76 shots and its hybrid on 40. Across all 82 configurations (122 comparisons, because Bell records are scored for each logical component and for failure of either), none of the paired intervals showed an improvement for CNN-plus-MWPM. They showed higher failure rates in 77 comparisons, typically by 1–2 pp (median 1.66 pp). The penalty appears mainly in the X-memory and Bell records, where phase errors and correlated errors are common. In the all-zero Z-memory records the difference was never resolved.

7.3.2 Repeated-check memory experiments

The repeated-check experiments used rotated patches at distances 3 and 5. Each circuit ran one to nine rounds of checks and then measured the data qubits. There are 23 combinations of distance, preparation and round count, with 4,096 shots each (94,208 in total). At distance 3, all-zero Z memory and plus-state X memory each use 1, 3, 5 and 7 rounds, giving eight conditions. At distance 5, the same two preparations each use 1, 2, 3, 5, 7 and 9 rounds, giving twelve more. The remaining three conditions use five rounds at distance 5: all-one Z memory, a fixed random Z-basis preparation and a fixed random X-basis preparation.

The ancillas are measured without reset, so interior detection events use the $m_t \oplus m_{t-2}$ rule from Section 2.3. Appendix C.5 reports how often the detectors fired in these records, by round and by position in the patch (Figures C.6 and C.7). DEM matching used median readout and two-qubit gate errors from the device calibration. The one-qubit gate error was set to one tenth of the two-qubit error.

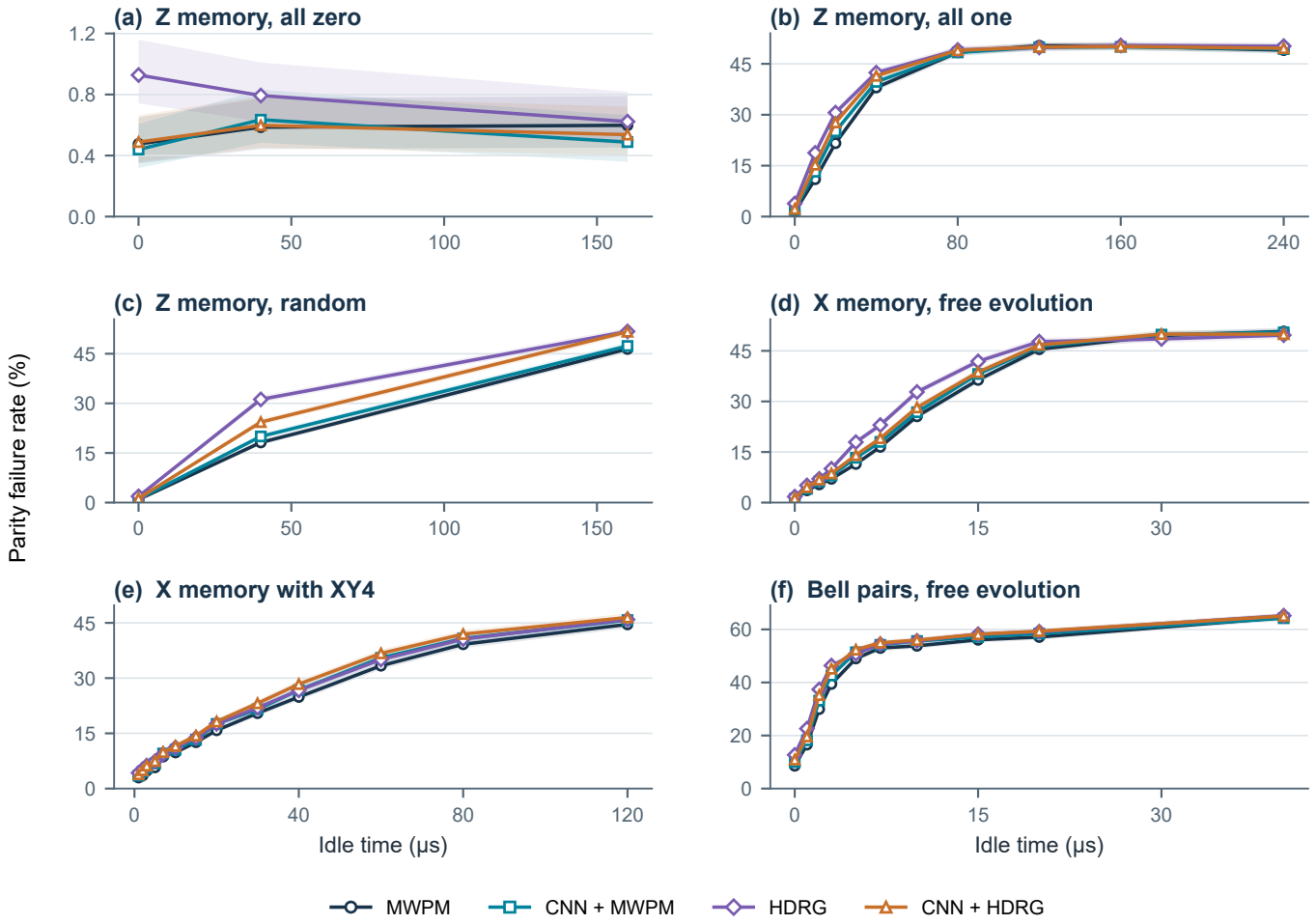


Figure 7.6: Decoder failure rates on final-measurement records from the IBM quantum processor

Failure rates from the same 8,192 shots per point, at distance 5. MWPM and HDRG run alone and after one integer CNN pass. Panels (a–c) use three Z-memory preparations, (d, e) compare free evolution and XY4 in X memory, and (f) counts failure of either logical component in the Bell experiments. The vertical scales differ. Shading gives 95% Wilson intervals.

Figure C.8 in Appendix C.6 shows the recorded data-qubit and ancilla placements for these distance-3 and distance-5 circuits. The physical connections determine which interactions the circuit can use; the network's tensor coordinates organise the resulting measurements for decoding.

Figure 7.7 shows the effect of one integer CNN pass in every condition. For HDRG, the paired interval indicated lower failure in 10 of the 23 conditions and higher failure in one, with the other differences unresolved. Eight of the 11 X-memory conditions improved, by up to 6 pp at distance 5. The PCM matching results were mixed: four conditions improved and six worsened. DEM matching showed no resolved improvement and 17 resolved increases. Each interval applies to one comparison, so these counts do not guarantee every improvement or increase across the full set.

Each row fixes the patch, number of rounds and preparation shown on the left, then compares three cleanup methods across the panels. Every panel uses the same shots. A point two percentage points to the left of zero means about two fewer wrong logical outcomes per hundred shots with the CNN. Moving to another row changes the experiment. Moving across the same row changes the cleanup method. This makes it possible to see a benefit for HDRG and a penalty for DEM matching on the same device records.

Four floating-point passes make the contrast more pronounced (Figure C.5 in Appendix C.5). At distance 5 with five rounds in Z memory, DEM matching alone fails on 807 of 4,096 shots and the four-pass hybrid on 1,094, an increase of 7.0 pp [5.6, 8.5]. In X memory, four passes reduce HDRG failures from 2,044 to 1,905, a gain of 3.4 pp [1.3, 5.5].

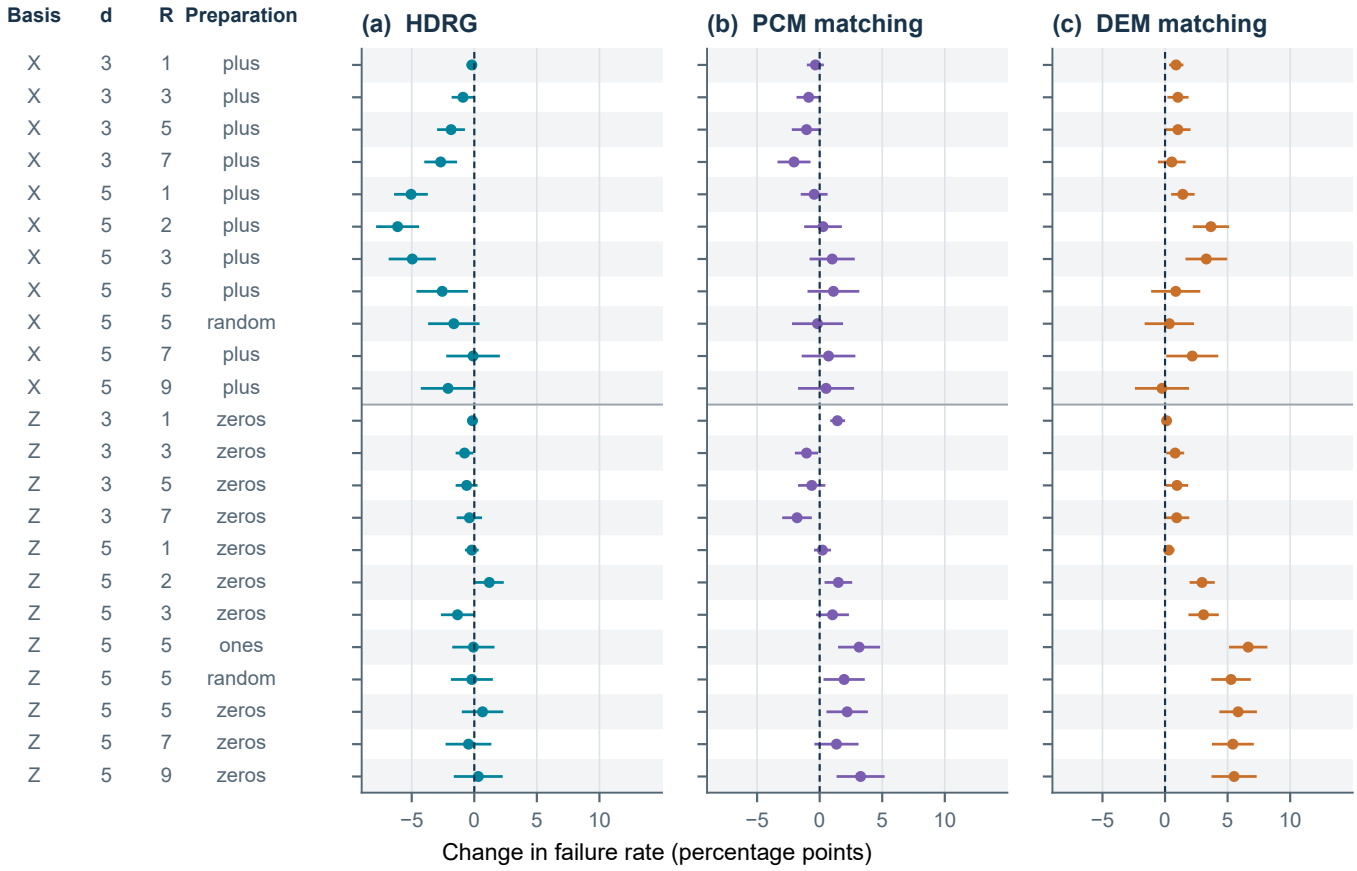


Figure 7.7: Effect of one CNN pass on repeated-check records from ibm_miami

Each row gives the result for one of 23 settings, with 4,096 shots per setting. The labels give the memory basis, distance d, number of rounds R and preparation. Left of zero, adding one integer CNN pass reduced failures relative to the same cleanup decoder alone. Right of zero, it increased them. The changes are in percentage points. Each bar spans the paired 95% confidence interval for that setting.

7.3.3 Detector removal before cleanup

The detector-removal test measures how much of the detector record remains for cleanup. It uses the first 512 shots from each of the 23 repeated-check conditions. Each shot received up to four passes, stopping when the residual was empty. Across the 23 conditions the number of detection events fell by 36.10% to 89.57%, with a median reduction of 88.78%. The five distance-5, five-round preparation families lie between 89.08% and 89.57%.

Figure 7.8 shows the all-zero Z-memory and plus-state X-memory conditions. The left panel counts detection events and the right counts shots with none left. Suppose ten shots each contain ten events. Leaving one event in each shot removes 90% of all events but clears no shots. Clearing nine shots and leaving the tenth untouched also removes 90% of events.

Even an empty residual can have the wrong logical parity if the correction crosses the patch along a logical path. This diagnostic counts removed events. The complete decoders above test logical recovery. Section 8.5 measures cleanup time separately.

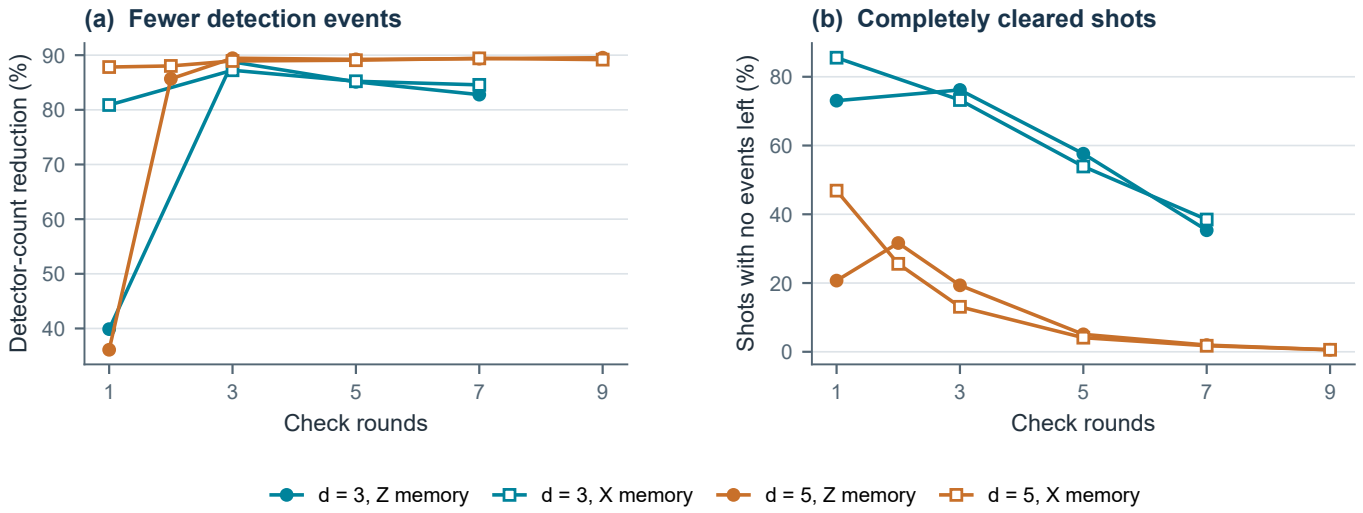


Figure 7.8: Detector removal on repeated-check records from ibm_miami

The first 512 shots from each all-zero Z-memory or plus-state X-memory condition, with up to four CNN passes. The passes stop when the residual is empty. (a) Detector removal is one minus the total residual count divided by the total original count. (b) The fraction of shots with an empty residual. Lines join the measured values. Each pass uses the same R measurement rounds.

7.4 Dependence on the trained network and cleanup decoder

For the two pretrained networks tested here, the clearest benefit occurs with the simpler cleanup baselines. The strongest matching baselines often perform better on the original syndrome. This is a result for these networks, their training and the tested noise conditions, rather than a general limitation of neural pre-decoding. Three features of the pipeline offer possible explanations for this pattern.

First, the cleanup decoder receives the residual syndrome after the CNN's correction. It can apply a compensating correction, so the CNN's choices are not physically irreversible. However, a cleared local pattern is no longer represented in the residual, and the cleanup decoder receives no measure of the CNN's confidence in that choice. Fewer residual events can therefore make cleanup faster without making the final answer more accurate. The 3D multi-pass comparison shows this distinction directly (Section 7.2.2).

Second, matching weights describe the original noise model. After preprocessing, the residual contains errors the CNN left unchanged or introduced, with a different distribution. The per-site reweighting in Section 7.1.2 did not consistently improve recovery. That test changed the probability at each site. It did not account for errors that tend to occur together.

Third, preprocessing can change the relationship between bit-like and phase-like errors. The 2D network predicts their corrections in separate output channels. Removing one component of a Y error leaves a different pattern from the correlated fault described by the original model. This is a possible explanation for the larger penalty with correlated matching, but that mechanism was not tested directly. Training and evaluating the network together with its intended cleanup decoder would help investigate it.

Recent work demonstrates the importance of that choice. Chamberland et al. report trained CNN pre-decoders that reduce logical error rates and combined CNN and matching time with uncorrelated PyMatching, including at distances 21 and 31 [42, Fig. 19]. Their networks and training labels differ from those used here. With correlated PyMatching,

they used a larger network and obtained lower logical error rates at distances 5, 9 and 13 [42, Sec. VI.2]. These results support evaluating the complete trained pipeline. The penalties measured in this thesis do not establish that a CNN cannot benefit a strong global decoder.

7.5 Implications for a hardware decoder

Under the tested conditions, one CNN pass improved HDRG and PCM matching in the circuit-noise simulation. Circuit-derived matching already recovered most windows, and adding the CNN could increase failures. Some HDRG settings improved on the records from the IBM quantum processor, while none showed a resolved gain for DEM matching.

The hardware runs one pass with a known computation cost. Additional passes sometimes increased failures. Paired intervals show how much difference the trials could resolve between the one-pass integer and floating-point rates.

The networks were trained on simulated noise. Training on device-like noise and evaluating the intended cleanup decoder would help to explain the difference between the simulated and device results. Appendix C.1 tests whether removing small weights, a possible hardware simplification, preserves quality after cleanup, and Appendix C.2 shows why agreement between local predictions cannot replace that test. Chapter 8 examines the time required for local prediction and global correction.

8. System integration and scaling

The board tests timed the CNN with its input already in memory (Section 4.6). In a quantum computer, the measurement bits must first reach the decoder, and the correction must reach the controller in time for the operation that needs it. This chapter compares those requirements with the FPGA timings, then examines how several FPGAs can share the calculation.

8.1 The path from qubit to correction

Figure 8.1 follows a measurement through a superconducting system. The readout resonator produces a microwave signal. The electronics amplify and digitise it, then combine the samples to decide whether the measured bit is zero or one. Systems such as QICK perform this processing on an RFSoC [12]. A live interface to the decoder would start here, with the classified bit, its check and round, and a flag indicating whether the readout was valid.

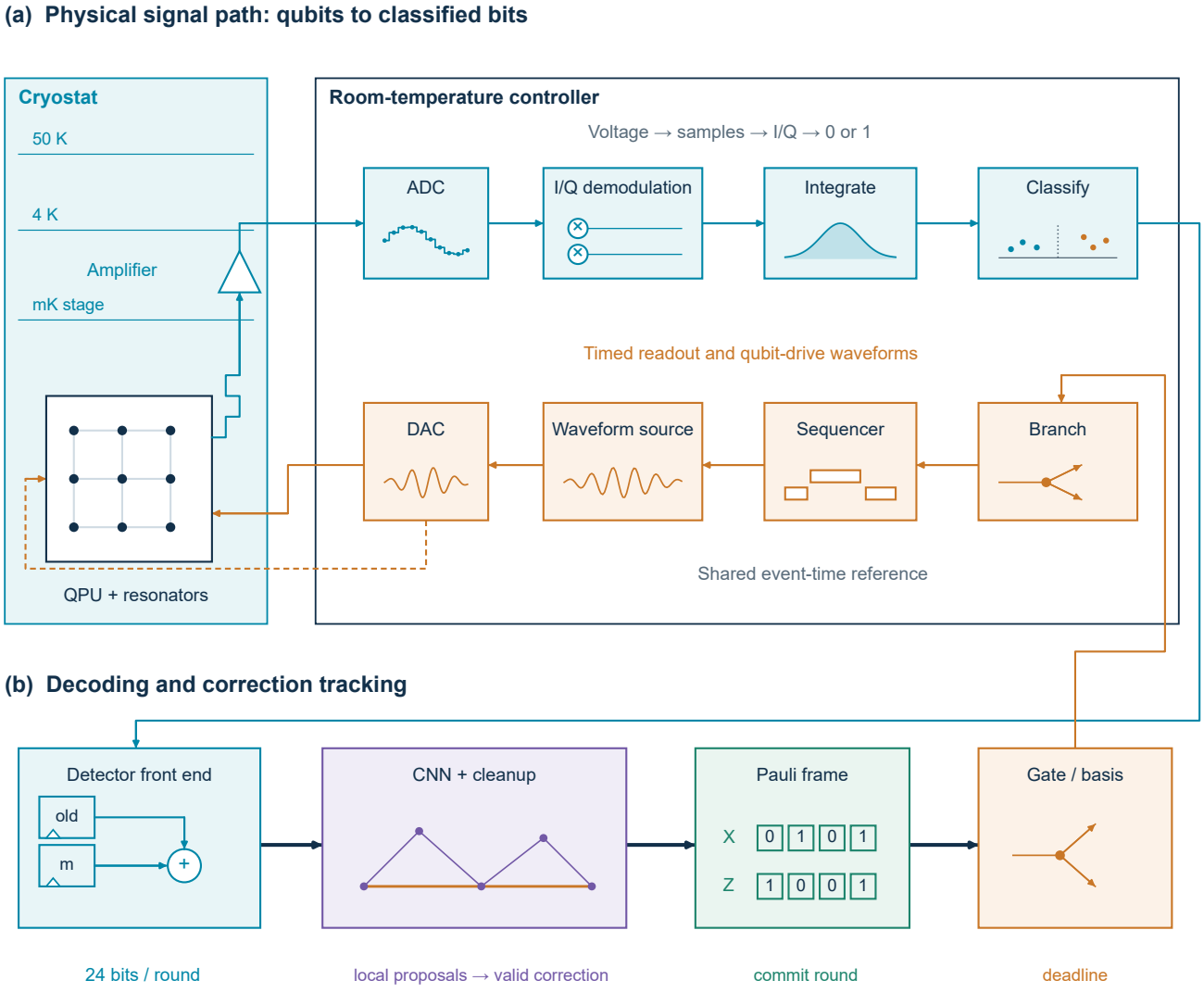


Figure 8.1: Superconducting readout and the path to a correction

(a) A superconducting readout chain: amplification, digitisation, demodulation, integration and classification into bits, alongside the control path that generates pulses. (b) The decoder side: a front end forms detection events from 24 check bits per round at distance 5. The CNN proposes a local correction, the cleanup decoder processes the remaining checks, and the Pauli frame records the combined correction before the operation that needs it. Positions and temperatures are schematic.

A binary outcome discards the confidence of the readout decision. A *soft-information* interface would also send a probability, allowing uncertain measurements to receive less weight during decoding. Caune et al. demonstrated this approach in an offline analysis of their FPGA experiment [13]. In simulated scaling tests, Google's AlphaQubit decoder also used soft readouts and information about leakage outside the qubit's two computational states [38]. The CNNs studied here accept binary inputs. Using readout confidence would require changes to their inputs and training, as well as a larger measurement record.

First, the decoder assigns each stabiliser measurement outcome to its check position and round, then compares it with the appropriate earlier outcome. In the circuits run on the IBM quantum processor the ancillas are reused without reset, so an interior detection event is $D_t = m_t \oplus m_{t-2}$ (Section 2.3). The CNN proposes local corrections. The global decoder processes the remaining checks. The controller can apply the combined correction or record it in its *Pauli frame*, so later measurements are interpreted correctly [6]. Correct round labels are essential: assigning a valid bit to the wrong round can create a false detection event.

Two outputs are useful in different settings. To maintain a Pauli frame for the full patch, the controller needs to know the proposed correction at each data qubit. If both the CNN and the global decoder propose an X correction at the same site, the two cancel. A pending X also reverses the interpretation of a later Z-basis measurement, so the controller can account for it without sending another pulse. For the memory experiments, it was sufficient to report whether to flip the measured logical parity. A controller must know which answer it is receiving and which round it belongs to.

Figure 8.2 places the steps on a time axis. One *shot* runs from preparation through R rounds of checks to a final measurement. A *window* contains the rounds used for one prediction. For example, 4,096 shots are 4,096 executions of the same circuit on the IBM quantum processor. The spacing between rounds within a shot sets the rate at which a live decoder must accept input.

8.2 Latency and throughput

Latency is the time from a round's measurements to its correction. *Throughput* is the rate at which the decoder processes rounds. It must sustain the arrival rate for continuous operation (Section 2.6).

Suppose a decoder accepts a new round every microsecond and returns its correction ten microseconds later. Once the pipeline fills, one answer arrives every microsecond. The controller must allow ten microseconds for each answer, but the input queue stays stable. If the decoder accepts only one round every two microseconds, that same input stream adds to the queue each time. A larger buffer allows a longer run before it fills.

The deadline is set by the next operation that needs the correction, which may occur several rounds later. Some operations can proceed while the update is pending. An operation chosen on the basis of an earlier measurement may need to wait. A connected experiment must therefore measure unusually slow responses as well as the average.

The centred 3D network uses four future rounds to predict a correction for the current round (Section 6.7). The waiting time for these measurements depends on the syndrome-extraction schedule. For the distance-5 experiment on the IBM quantum processor, the durations are those estimated by the compiler from the operations scheduled on *ibm_miami*. With the same all-zero preparation, the one-round circuit takes 6.472 μs and the nine-round circuit takes 39.048 μs.

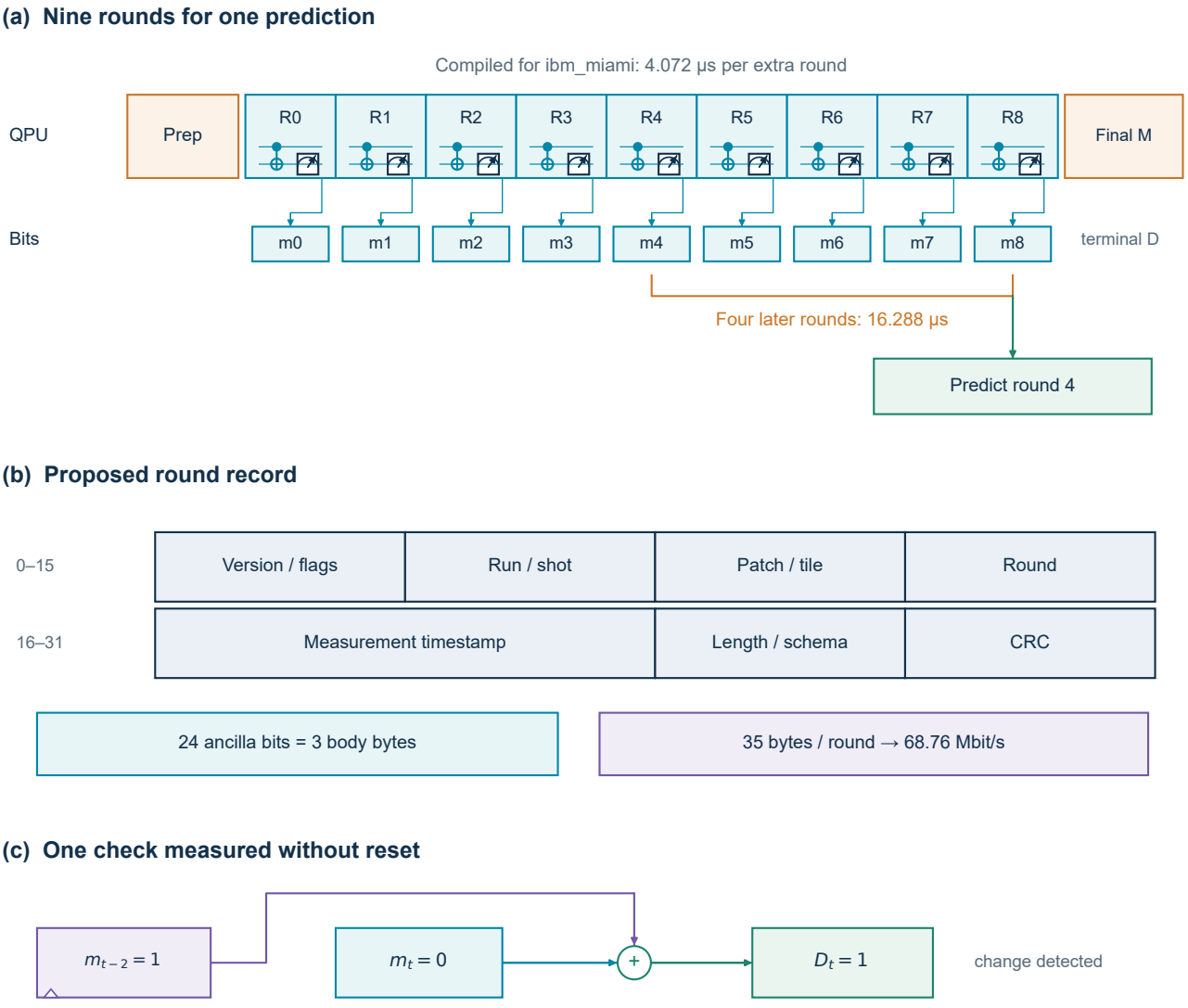


Figure 8.2: Check records and the context for one 3D prediction

(a) A prediction for the middle of nine round records needs four later rounds. At the 4.072 µs round increment compiled for the IBM quantum processor, the wait from the middle record to the last is 16.288 µs. (b) A proposed 32-byte header with identity, timing and an error check, plus three bytes of distance-5 check data, gives 35 bytes per round. (c) Without ancilla reset, an interior detector compares a measurement with the one two rounds earlier.

The eight added rounds account for 32.576 µs, or **4.072 µs per added round**. The circuits with two, three, five and seven rounds give the same increment, as do the X-basis circuits prepared in the plus state. Taking the difference removes the fixed preparation and final-readout time.

Figure 8.3 shows the compiled durations for all six round counts. On the schedule compiled for *ibm_miami*, round $t + 4$ ends 16.288 µs after round t ends. The decoder can already work on earlier measurements as they arrive. The same four-round wait is 4.4 µs at Willow's measured round period of 1.1 µs [8]. The interval for the IBM QPU is a compiler estimate, whereas the Willow round period is a published experimental measurement. Measuring the interval between classified records on that processor would require timestamps at the controller.

8.3 Quantum-computer interfaces and timing

The connection to a decoder depends on how the quantum computer measures its qubits. In a superconducting system, microwave readout is digitised and classified into bits. In trapped-ion and neutral-atom systems, the electronics infer

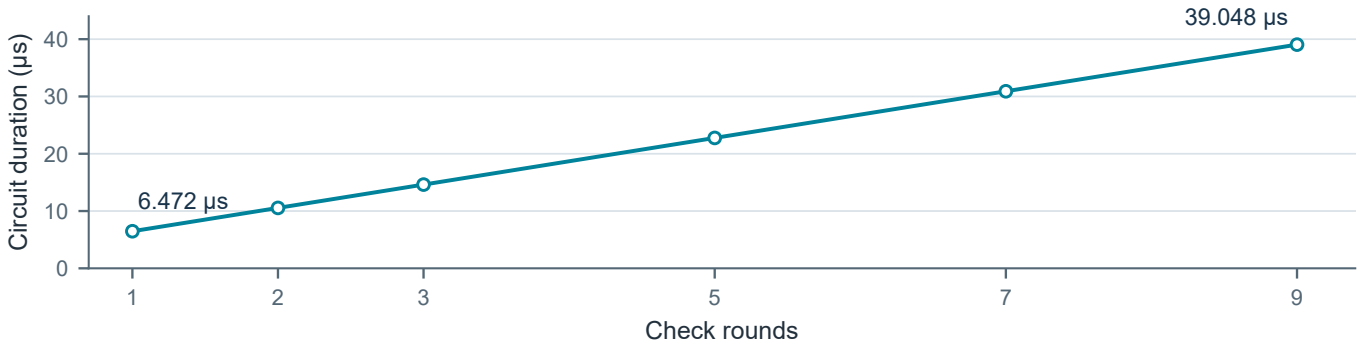


Figure 8.3: Circuit durations compiled for ibm_miami

For distance-5 Z memory prepared in the all-zero state, the compiler schedules 6.472 µs for one check round and 39.048 µs for nine. Across the six circuits shown, each additional round adds 4.072 µs.

the state from the light they detect. A neutral-atom system may also need to report whether an atom is missing. Figure 8.4 shows where these different readout paths could meet a classical decoder. The common starting point is a labelled measurement record, after signal classification.

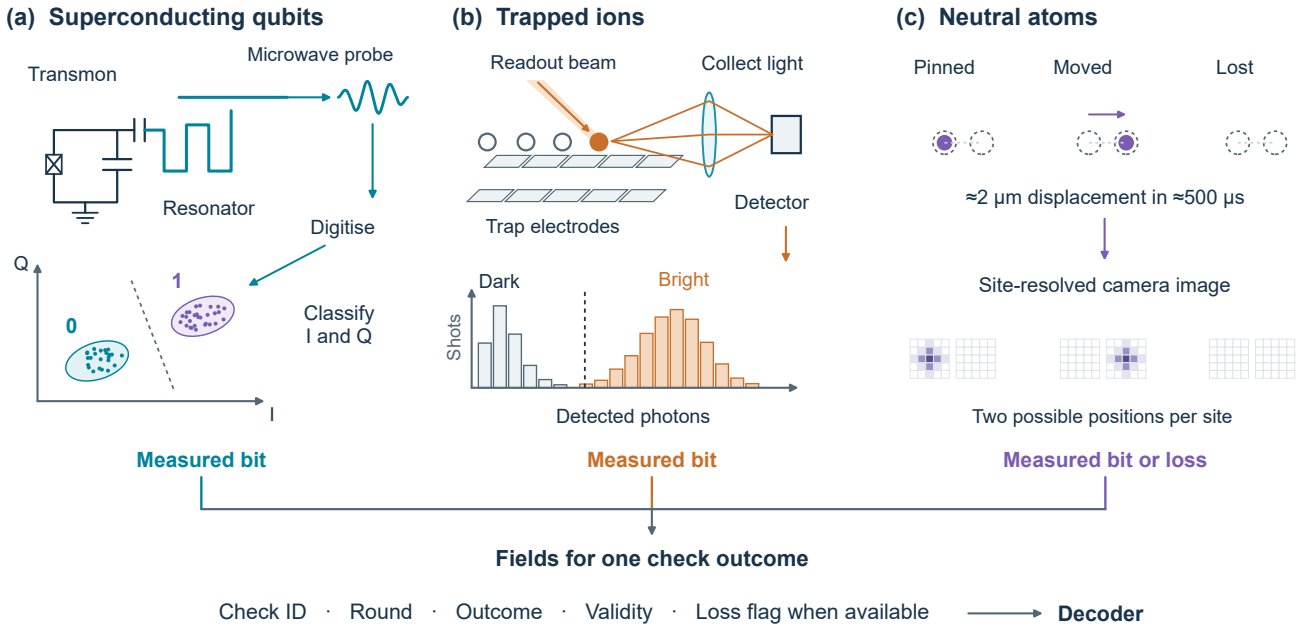


Figure 8.4: From qubit readout to a decoder input

(a) A superconducting qubit shifts the response of a nearby resonator. The electronics measure two components of the returning microwave signal, I and Q, and classify each reading. Willow completed a full check round in 1.1 µs [8]. (b) A detector counts light scattered by an illuminated ion. A count threshold separates bright and dark readings. In one trapped-ion controller, distributing a conditional result added less than 250 µs in the worst case [47]. (c) A tweezer moved atoms in one spin state by about 2 µm in 500 µs. A camera then distinguished atoms at the original position, atoms at the displaced position and empty sites [48]. The signals and images are illustrative. Any one of these readouts could supply the proposed record. A loss flag lets the decoder distinguish a missing atom from a measured zero.

Consider an empty site in a neutral-atom register. A missing atom cannot produce a valid measurement of zero. Recording a loss flag lets the decoder distinguish the empty site from one observed in state zero. Before using either CNN on another platform, its input, cleanup graph and accuracy would need to be checked for the records that platform produces.

Willow illustrates the difference between round time and response time (Section 3.5). Its checks ran every 1.1 µs, yet its distance-5 decoder took 63 ± 17 µs from the last syndrome reaching the decoding software to the returned

correction [8]. On Ankaa-2, Caune et al. measured 9.6 µs from the end of the last readout to a conditional gate after nine rounds. The FPGA decoder used 6.5 µs and communication and control used 3.1 µs [13].

In the tests reported here, the fastest Arty core took 1.154 ms for one loaded distance-33 2D frame. A larger resident core took 2,180 simulated cycles, or 21.8 µs at the clock it met after routing. For a complete distance-33 2D frame, the 64-FPGA model predicts 0.969 µs. Section 8.9 also gives the interval between successive slices of a 3D volume. All three times stop at the CNN prediction. The controller would still need to deliver the measurements and receive a correction after cleanup. A distance-5 volume with five rounds took 2.354 ms on the Arty, about 580 of the round increments compiled for *ibm_miami*.

On a ten-qubit trapped-ion system, Ryan-Anderson et al. completed an adaptive QEC cycle in less than 200 ms. That time includes ion transport, cooling, gates, readout and feedback [47].

For neutral atoms, Bluvstein et al. report a 4.45 ms coherent extraction round with ancilla readout later [48]. In a separate toric-code experiment, extraction took about 80 or 150 ms, depending on code size, before measurement and atom replacement [49]. For a decoder, the available time begins when the classified bits are ready and ends at the first operation that needs their correction.

8.4 Input data rates

The controller must also supply measurement data at the required rate. A rotated distance-d patch measures $M = d^2 - 1$ checks per round. With a round period τ, P patches and a 32-byte header per patch-round carrying identity, timing and a transmission error check, the raw and record rates are

$$B_{\text{bits}} = \frac{PM}{\tau}, \qquad B_{\text{record}} = \frac{8P\,[32 + [M/8]]}{\tau}.$$

At distance 5, one patch supplies 24 check bits per round. Using the 4.072 µs round increment compiled for the IBM QPU gives 5.9 Mbit/s. Adding the example header raises the rate to 68.8 Mbit/s because the header is much larger than the three bytes of check data. At distance 33, each patch supplies 1,088 bits. A 1.1 µs round therefore requires 989 Mbit/s for the check bits, or 1.22 Gbit/s including the header. The same record sent once per millisecond would need 1.34 Mbit/s.

One way to save bandwidth is to send several rounds under one header. For instance, at distance 5 a single round needs three bytes of check data. Four separate records would carry $4(32 + 3) = 140$ bytes. Combining the four rounds in one record uses 44 bytes. If the transmitter waits for the fourth round before sending, the first is already three rounds old. In general the earliest round waits up to $(g - 1)\tau$ when g rounds share a header.

The 115,200-baud UART on the Arty was adequate for checking complete frames. A controller sending an uninterrupted stream at the rates above would need a faster link.

8.5 Cleanup time with and without the CNN

CPU cleanup. If the CNN clears enough checks, it can reduce the time a global decoder spends on the rest. Each cleanup decoder was tested on 2,048 stored syndromes, both before and after one integer CNN pass. After 128 warm-

up calls, the timing test ran each case three times in random order on the same core of an Intel Core Ultra 9 386H. The times include only the decoder call (Table 8.1).

Table 8.1: CPU cleanup time in microseconds per record, before and after one integer CNN pass ($p = 0.13$ in 2D, $p' = 0.005$ in 3D), and the ratio of the means. In the 3D rows, d is the code distance and R is the number of rounds of check measurements in one memory circuit. Each pair gives the mean and the observed 99th percentile (p99). Matching uses PyMatching, with the parity-check graph for PCM. The 2D HDRG code is compiled with Numba. The 3D HDRG code runs in Python.

Workload	Cleanup	Before: mean / p99	After: mean / p99	Speed-up
2D, $d = 33$	MWPM	245.6 / 415.7	92.4 / 173.4	2.66
2D, $d = 33$	HDRG	63.3 / 106.3	23.7 / 35.8	2.67
3D, $d = R = 5$, X	DEM matching	5.29 / 12.00	3.42 / 7.90	1.55
3D, $d = R = 5$, Z	DEM matching	5.10 / 11.56	3.54 / 8.20	1.44
3D, $d = R = 5$, Z	PCM matching	5.22 / 10.50	4.08 / 7.90	1.28
3D, $d = R = 5$, X	HDRG	119.2 / 212.2	61.6 / 190.6	1.94
3D, $d = R = 5$, Z	HDRG	115.9 / 212.5	66.7 / 191.8	1.74
3D, $d = R = 33$, X	DEM matching	1,934 / 2,692	1,214 / 2,055	1.59
3D, $d = R = 33$, Z	DEM matching	1,947 / 2,817	1,175 / 1,916	1.66
3D, $d = R = 33$, X	Correlated DEM	5,989 / 7,378	2,840 / 4,455	2.11
3D, $d = R = 33$, Z	HDRG	10,155 / 14,366	2,374 / 3,027	4.28

The 2D software returns a correction for each of the two error components. The 3D software returns a decision about the measured logical parity. The timer stopped when the call returned. Constructing the matching graph, converting measurements to detectors, running the CNN, forming its residual and returning a controller message lie outside that interval. Reading across a row compares the same software before and after the CNN. Reading down a column also compares different implementations.

Mean CPU cleanup time fell in all 28 timed conditions, of which Table 8.1 shows eleven. The repository holds the complete timings. The smallest speed-up was 1.28 for distance-5 PCM matching and the largest was 4.28 for distance-33 HDRG, both in 3D Z memory. The saving must also exceed the time taken by the CNN itself. At 2D distance 33 the MWPM saving is 153 µs, while the fastest Arty core takes 1,154 µs, so adding this core increases the total time. A faster implementation would need to fit inference, the residual update and added data movement inside that saving. Its decoding quality also depends on the cleanup decoder, as Chapter 7 showed.

The three-lane resident core in Section 8.7 offers a different timing budget. Its 21.8 µs inference plus 92.4 µs mean MWPM cleanup totals 114.2 µs, compared with 245.6 µs for MWPM alone. That leaves 131.4 µs for transfer and residual construction. With HDRG, 45.5 µs of processing against 63.3 µs leaves only 17.8 µs. These budgets combine simulated FPGA cycles at a routed clock with separate CPU calls. A connected measurement would show how much of each saving remains after the transfers.

For a 3D example, consider a distance-5 X-memory frame with DEM matching. The Arty takes 2,353.7 µs for the CNN. The 194-byte request and 150-byte reply, including commands, the cycle counter and framing, would take 1.6 and 1.2 µs over an assumed 1 Gbit/s link, and mean CPU cleanup takes 3.4 µs. That gives 2,359.9 µs before input conversion, residual construction and controller delivery. This example starts after the final classified measurement of a finite experiment. The four-future-round wait in Section 8.2 applies to a continuing stream.

Distance-3 lookup. RTL simulation measured the time from the CNN's last prediction to the combined correction. Once the input frame was in memory, the 2D CNN took 1,671 cycles. Reading its predictions, forming the residual, looking up the global correction and combining the masks took another 17 cycles. The connected design passed internal timing at 100 MHz on the Arty target. At that frequency the 17 added cycles take 170 ns, giving a total of 16.88 µs from a preloaded frame. Both lookup modes matched the integer software reference on each of 32 inputs, giving 64 correct results. The lookup itself took only one cycle. Reading the 13 data sites from the CNN's output memory accounted for most of the added time.

Direct calculation also gives short latencies on small patches. The distance-3 minimum-weight search took 210 ns, and HDRG took at most 90 ns at distance 3 and 270 ns at distance 5, all at a routed 100 MHz with the residual ready.

Connected HDRG through distance 33. For larger patches the banked CNN was connected to two HDRG engines that scan one row at a time. At distance 33, reading the predictions and transferring the residual alone took 2,115 cycles.

Each of the 32 distance-33 frames at 13% physical error probability was used twice. HDRG first received the original checks and then the checks left by the CNN. The search was faster after the CNN on every frame. Its median fell from 4,746 to 1,163 cycles.

The connected circuit must also read the predictions and combine the corrections. Counting from the last CNN prediction to the combined correction gave a median of 3,279 cycles. At the 100 MHz clock the design met after routing on a Virtex UltraScale+ VU13P, that is 32.79 µs. The CNN itself required 150,369 cycles, so the median for the complete loaded frame was 1.536 ms. HDRG alone finished sooner on all 32 frames. The CNN can improve the correction, as Chapter 7 showed. Appendix B.8 shows the circuit (Figure B.3) and the measurements across distances (Table B.5).

Since inference took longer than the search time it saved, the next sections test whether more CNN calculations can run in parallel, first in one FPGA circuit and then across several.

8.6 Partitioning the convolution across FPGAs

A larger code produces a larger map. In the multi-FPGA simulation, each modelled FPGA runs one decoder circuit, called a worker. The map is split into rectangular tiles, and each tile is assigned to one worker, which computes all its output values. A worker still needs some input from outside its block. This strip of additional input is its *halo*. An 11×11 filter reaches five positions beyond an output in either direction. After three such layers the outermost input that can affect an output is fifteen sites away. A 2D worker can receive all fifteen additional sites at the start, or exchange a five-site strip after each layer (Figure 8.5). The four $3 \times 3 \times 3$ layers in the 3D network reach four cells in space and four slices in time.

At an internal tile edge, a worker needs its neighbour's values. At the outside of the full image or volume, each layer needs zero padding. Applying zero padding at an internal edge would change the convolution. Padding is needed only at the full map's boundary, at every layer. If a worker computes values beyond the full map as though they were ordinary sites, a bias can make them nonzero and change predictions near the edge. With the correct rules, all 102

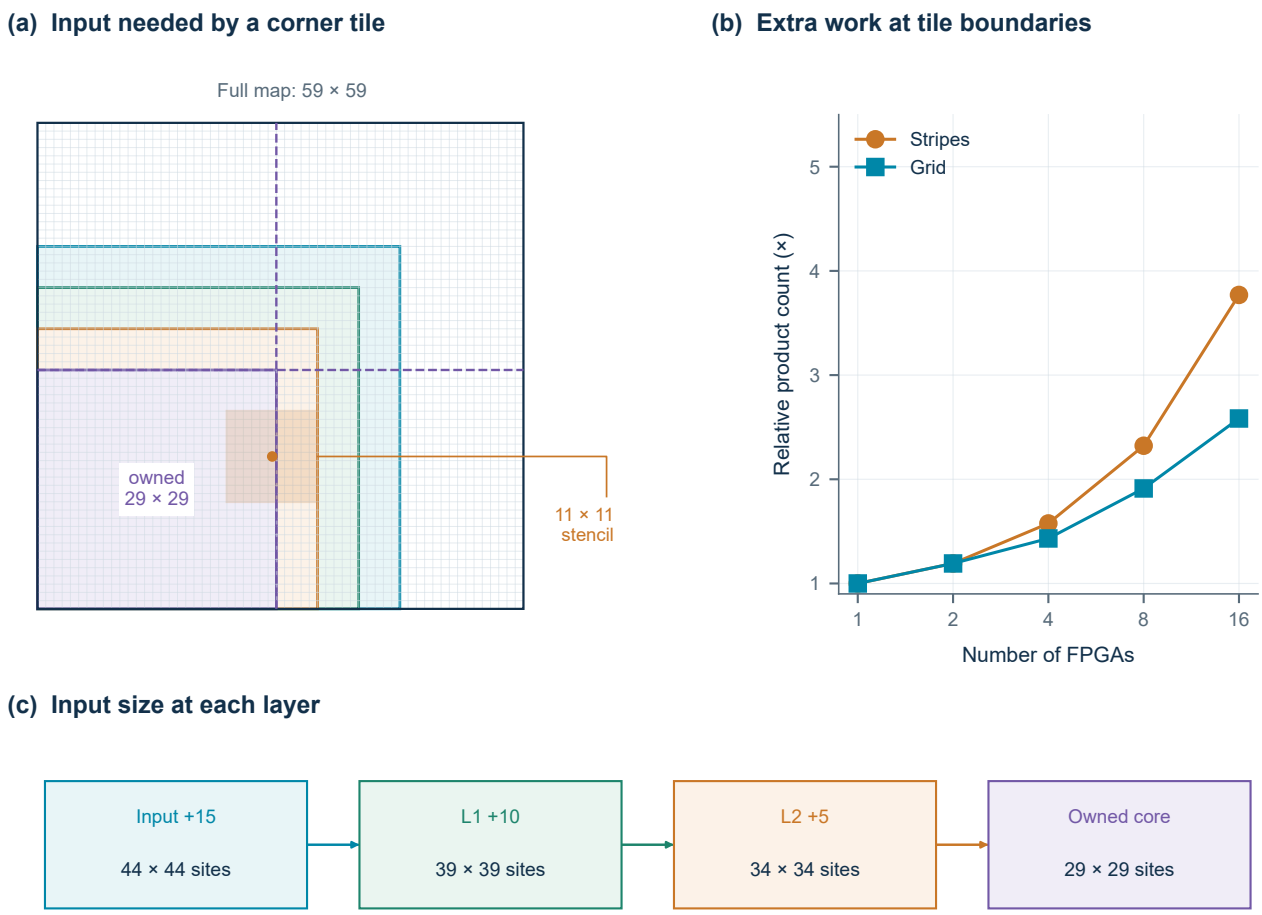


Figure 8.5: Owned regions, halos and the cost of overlap

(a) Four workers split the 59×59 map of a distance-30 patch. The highlighted worker owns 29×29 outputs and needs neighbouring inputs for its 11×11 kernel. (b) The total multiplication count relative to one worker is lower for square tiles than for stripes. (c) Working backwards, the corner tile needs 34×34 and 39×39 hidden regions and 44×44 inputs, clipped at the map's outer edges. An interior tile needs a halo on both sides of each axis.

software tests of 2D and 3D partitions reproduced the undivided integer calculation, including unequal tiles and up to 64 workers (Appendix B.6).

For example, four FPGAs could each compute one quarter of a 3D network's output plane. Each worker would receive enough neighbouring inputs to run all four layers. A collector would assemble the predictions at their original coordinates. An error chain can cross from one tile into another, so the cleanup decoder would use the remaining checks across the whole patch.

Sharing hidden values after each layer is another option. These transfers are wider than the incoming checks. In the 2D hardware, for example, the nine channels at one layer-two position occupy 189 bits. A neighbouring worker needs all nine to compute the next layer, even though the final prediction at that site has only two bits. A link sized only for the incoming check records may have too little capacity for these exchanges. Appendix B.7 gives the record format and counts.

8.7 Overlapping layers in a resident worker

A single worker was used to test whether starting each layer as soon as its required rows were ready reduced inference time. The *resident worker* keeps its whole input in local memory and uses fixed-weight arithmetic, in which the trained coefficients are built into the logic. It is a different design from the resource-shared Arty cores of Chapter 5. In its *serial* schedule each layer waits for the previous one to finish. In its *row-ready* schedule a layer reads a row as soon as the producing layer has written every value that row needs, and pauses otherwise. This applies the readiness check of Section 5.4 to whole rows. Both schedules use the same arithmetic.

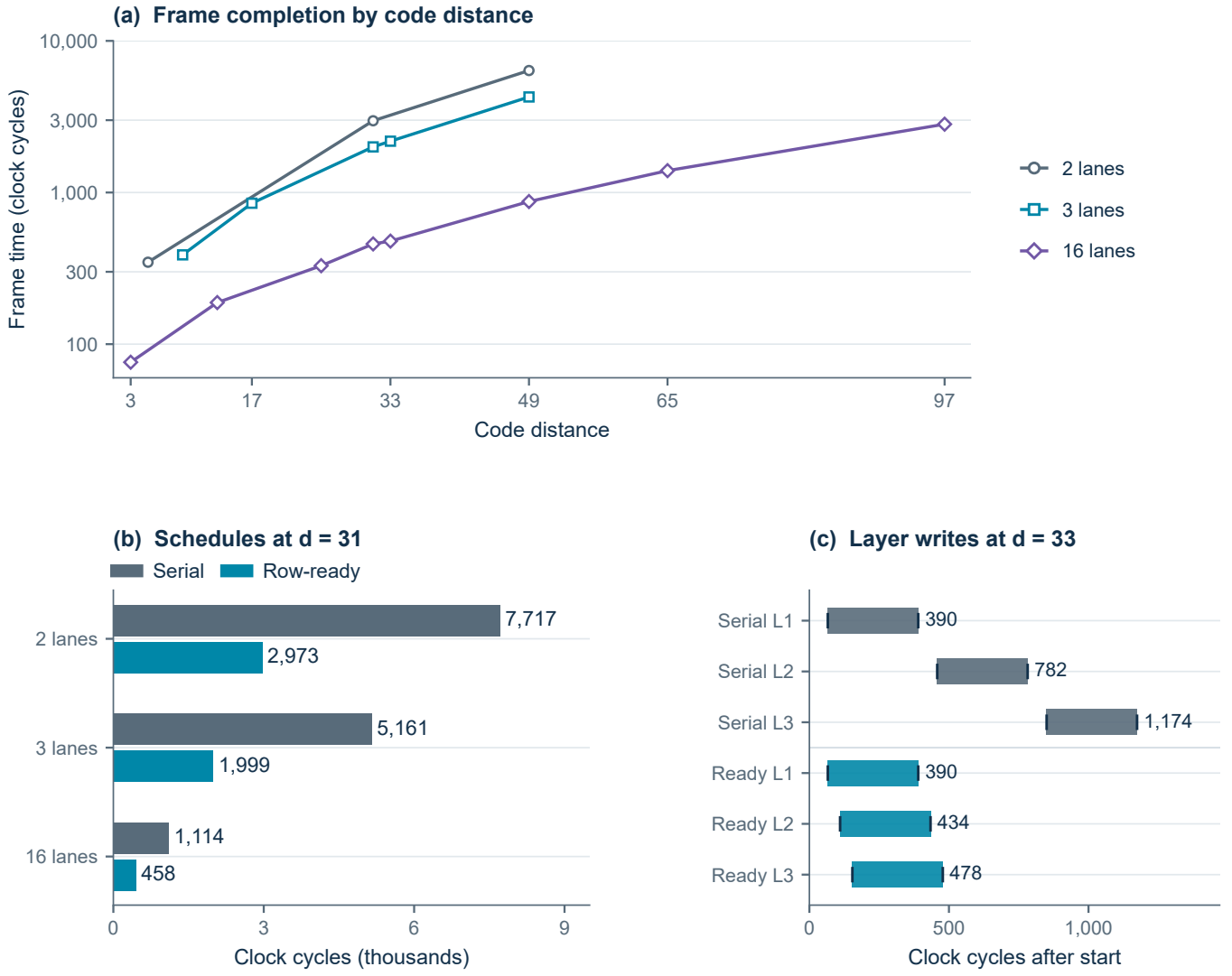


Figure 8.6: Overlapping layers in the resident worker

(a) RTL cycle counts for two, three and sixteen spatial lanes up to distance 97 (logarithmic axis). (b) The serial and row-ready schedules use the same arithmetic at distance 31, the only distance tested with both schedules at all three lane counts. (c) At distance 33 with sixteen lanes, the span of each layer's output writes: with row readiness the layers overlap and the run finishes at cycle 478 instead of 1,174.

At distance 31 with three spatial lanes, row readiness reduced the time from 5,161 to 1,999 cycles; at distance 33 with sixteen lanes, from 1,174 to 478 cycles, 2.46 times faster (Figure 8.6). The first layer finishes at cycle 390 in both schedules; with row readiness the later layers have already completed most of their work by that cycle. In RTL simulation the sixteen-lane worker reached distance 97 in 2,822 cycles, and a three-lane distance-33 worker restarted

1,000 times on alternating preloaded inputs, finishing every run in 2,180 cycles with all intermediate values checked (Appendix B.6).

In that repeated test, a new start was accepted every 2,181 cycles. The one-cycle difference separates the time to finish a frame from the spacing between starts. Both input slots were already loaded, so this tests repeated use of stored inputs, not the arrival of fresh quantum measurements.

To estimate latency from these cycle counts, the circuit must meet timing at the chosen clock. The three-lane distance-33 worker routed on a Versal Premium VP1902 and met internal timing at 100 MHz, so 2,180 cycles correspond to 21.8 µs from start to the last result, excluding input loading and readback. The worker did not meet its 250 MHz target, and the same design failed to route on a Virtex UltraScale+ VU13P (Appendix B.5). Compared with the Arty's 1,154 µs at distance 33, this is a 53-fold reduction, achieved using fixed weights and about 832,000 LUTs, thirteen times the Arty's entire LUT capacity.

A connected RTL test linked this worker to the two HDRG engines and repeated the distance-33 experiment from Section 8.5. A faster CNN alone would still leave 2,113 predictions to read from memory, one per cycle. Keeping the earlier readback path and allowing 2,180 cycles for inference gives a projected median of 5,459 cycles. Only four of the 32 frames would finish sooner than with HDRG alone.

The worker already produces up to three predictions per cycle. The connected design saves the two correction masks as the predictions arrive, so cleanup can start without a second pass through memory. In the connected RTL simulation, the median time from a loaded input to the combined correction fell to 3,346 cycles, compared with 4,746 for HDRG alone. The combined circuit finished sooner on all 32 frames and on 32 more drawn at the same noise rate. All local corrections, remaining checks and final masks agreed with the software reference. The 100 MHz timing result applies to the worker alone. The clock frequency of the complete circuit remains to be checked after routing.

The advantage depends on the error rate, because inference takes 2,180 cycles regardless of the input. A separate set of controls used four random frames at each of four error probabilities (Table 8.2). At 1% and 5%, HDRG alone finished first on every frame. At 13% and 20%, the connected circuit finished first on all eight. These small samples show that either design can be faster. They do not establish a precise crossover. On an empty frame, HDRG alone took only 33 cycles, while the CNN and HDRG together took 2,216.

Table 8.2: Clock cycles from a loaded distance-33 frame to its correction in RTL simulation. Each range spans four frames at the stated physical error probability.

Error probability	HDRG alone	Resident CNN and HDRG	Finished first
1%	246–338	2,216	HDRG alone, 4 of 4
5%	1,162–1,614	2,239–2,541	HDRG alone, 4 of 4
13%	4,556–6,149	3,410–4,281	CNN and HDRG, 4 of 4
20%	6,669–8,545	4,734–5,424	CNN and HDRG, 4 of 4

8.8 Distributing the computation

The simulation models 2D workers with local memories and input and output queues. Figure 8.7(a) shows how four and sixteen workers divide the same distance-33 map. The switch sends each value to the workers that need it. In one

four-worker trace, the layer-two value at row 20, column 32 belonged to worker 0, immediately inside its right edge. Both worker 0 and worker 1 received the nine channels at that site for the next convolution. After the last layer, the host collects each two-bit prediction once.

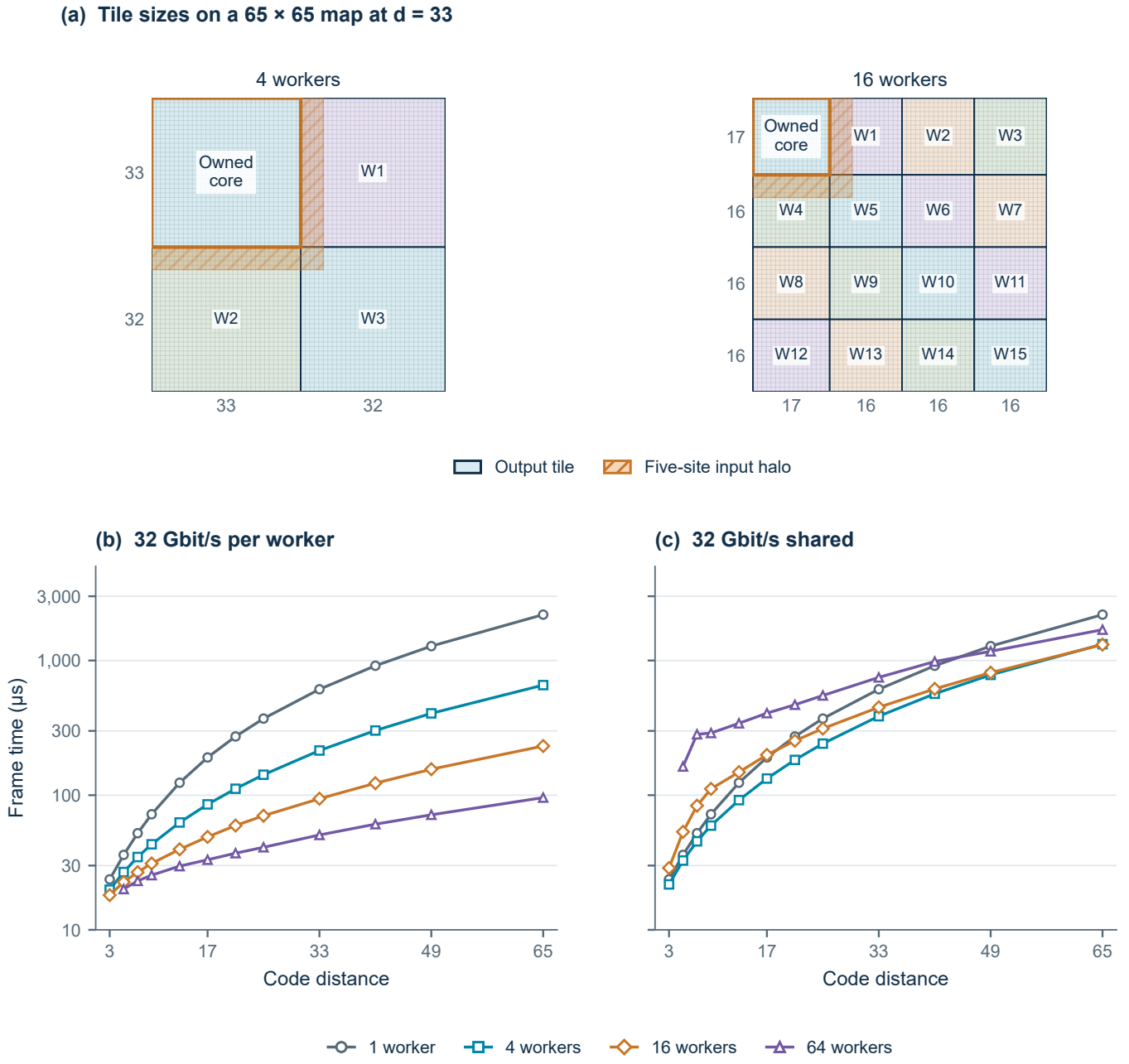


Figure 8.7: 2D worker partitions and simulated frame time

(a) Four or sixteen workers divide the same distance-33 map of 65 × 65 sites. The highlighted worker owns the outlined outputs and reads five extra sites beyond each internal edge for one convolution. (b) Each worker has 32 Gbit/s in each direction. (c) All workers share 32 Gbit/s per direction. Both plots show RTL time from the first accepted input to the last collected prediction, on a logarithmic scale. Every tested partition reproduced the whole-map integer result.

Each port accepts one 320-bit record every 10 ns, or 32 Gbit/s per direction. These internal messages carry values between workers; they differ from the controller records in Section 8.4. Each frame was timed from the first accepted host input to the last returned prediction. The measured time includes loading, exchange and collection, but excludes global cleanup. Across twelve distances from 3 to 65, 618 executions with one, four, sixteen or 64 workers reproduced the whole-map reference on 47.6 million checked sums.

A shared-path design used one communication path and scanned the whole map for every worker. On a 17×17 map, sixteen workers took 120.1 µs and four took 61.2 µs: more workers produced more overlapping traffic without more network capacity. The balanced design gives each worker its own ports and sends it only the input it needs. Table 8.3 shows the result at distance 33 under three communication budgets.

Table 8.3: Distance-33 completion time in microseconds from RTL simulation, with 10 ns clock periods and four-record queues. Budgets apply to each transfer direction. Bold marks the fastest worker count in each column.

Workers	32 Gbit/s per port	32 Gbit/s in total	128 Gbit/s in total
1	611.7	611.7	—
4	214.6	**385.8**	214.6
16	94.1	449.3	**149.6**
64	**50.6**	749.1	209.8

The repeated one-worker entries share one run, as do the four-worker per-port and 128 Gbit/s entries. There is no separate one-worker 128 Gbit/s test because its single port remains capped at 32 Gbit/s.

Figure 8.7(b, c) shows how the communication budget affects the choice of worker count. With separate 32 Gbit/s ports, one worker took 2,191 µs at distance 65 and 64 workers took 95.9 µs. With 32 Gbit/s shared, four were fastest at every sampled distance up to 49, and 64 were always slower than sixteen. At distance 33, raising the shared budget to 128 Gbit/s made sixteen fastest. Adding workers beyond sixteen increased completion time by 40% (Table 8.3). Buffering also affected the result. With per-worker ports at that distance, doubling the queues from four to eight records reduced the sixteen-worker time from 94.1 to 74.9 µs.

At distance 65 under the fixed 32 Gbit/s budget, sixteen workers took 1,311.55 µs and four took 1,321.46 µs. Sixteen were faster by about 0.75%. For this workload and communication policy, the advantage switched between the two sampled distances, 49 and 65.

The reason is visible in the transfer counts. At distance 33, dividing the same frame among 64 workers instead of four raises input deliveries from 5,625 to 18,225. Hidden values delivered to a different worker rise tenfold, from 2,800 to 28,000, while the final map still has 4,225 positions. Separate ports can serve many of these transfers in parallel. With a fixed total bandwidth, they compete for the same slots (Table B.3 in Appendix B.7).

One complete interior worker from the sixteen-worker distance-33 partition was placed and routed on a VU13P. It met internal timing with a 62.5 MHz worker clock and a 100 MHz switch clock (Table B.2). Simulating all sixteen workers at these clock frequencies reproduced the whole-map result in about 134 µs per finite frame, including after an interrupted calculation. Only this interior worker geometry was placed and routed.

8.9 Estimates for larger FPGAs

RTL simulation tested the resident and distributed workers. A resource model extends the comparison to five larger AMD parts using their published capacities: the Virtex UltraScale+ VU9P, VU13P and VU19P, and the Versal Premium VP1802 and VP1902 [59, 60]. Each configuration sets how many products to compute together, how many sites to process in parallel and how to supply the required memory reads.

The model reserves some capacity for routing and control. It rejects a design using more than 60% of the LUTs or flip-flops, 75% of the DSPs or 70% of block memory. Each device runs at an assumed 250 MHz. Each link direction can carry 400 Gbit/s, with 90% useful payload, 128-bit headers and 50 ns per link crossed. The estimates start with the inputs in memory. These starting conditions differ from those of the distributed RTL tests and a live quantum stream.

For 2D on VP1902 devices, a distance-9 frame takes a predicted 0.48 µs on one device and 1.80 µs on 64: for a small workload the additional communication outweighs the saved computation. At distance 33 the comparison reverses, from 4.66 µs on one device to 0.97 µs on 64, and at distance 97 from 31.8 µs to 2.37 µs (Figure 8.8). If the total network capacity is instead fixed at 3,200 Gbit/s, sixteen devices finish sooner than 64 at distance 33 (2.16 µs against 4.96 µs). As in the distributed RTL tests, the fastest device count depends on the communication budget.

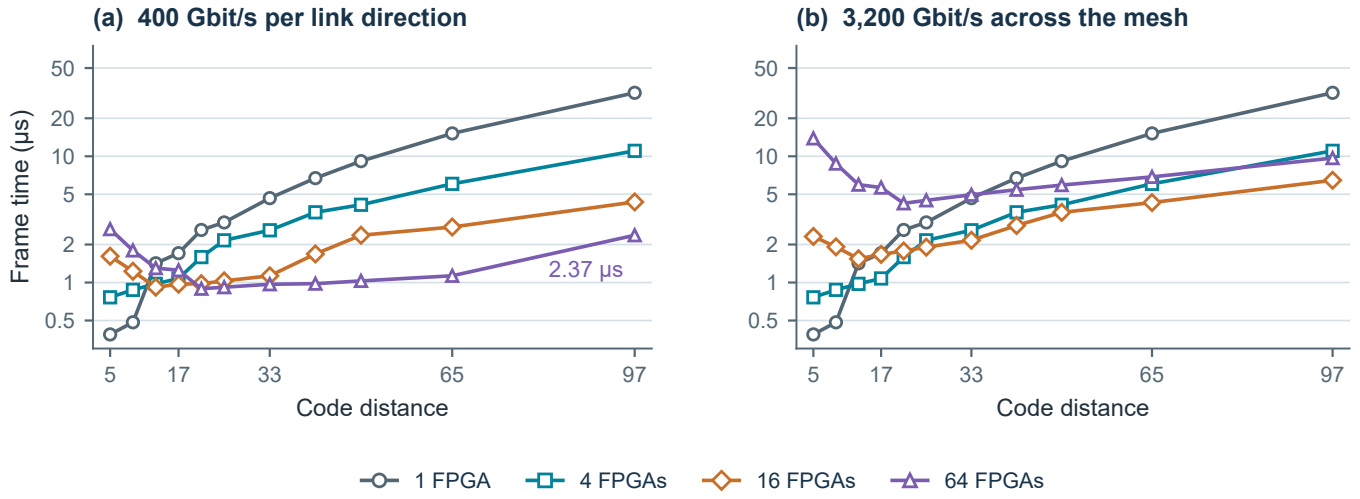


Figure 8.8: Model estimates for one 2D frame

Predicted time for one loaded frame on VP1902 devices at 250 MHz. (a) Each link direction has 400 Gbit/s. (b) The mesh shares a fixed 3,200 Gbit/s. Each point uses the fastest configuration found within the resource limits at that distance. With 64 devices in (a), the curve stays shallow between distances 21 and 65. At distance 97, larger tiles and boundary exchanges raise the time to 2.37 µs. The model excludes collection, cleanup and feedback.

At distance 33 with 33 rounds, the fastest 64-device VP1902 configuration finishes a loaded 3D volume in a predicted 29.7 µs. Its predicted interval between slices is 0.712 µs (Figure 8.9). A streaming implementation that sustained that interval could match the rate of Willow's 1.1 µs rounds. The same model needs 1.15 µs per slice at distance 49 and 1.87 µs at distance 65. Under these assumptions it crosses the Willow interval between distances 33 and 49 and stays below the 4.072 µs compiled round increment at all three distances. A hardware stream would still need to be tested at those rates.

The distance-33 conclusion depends on both clock and link speed. Changing only the clock from 250 MHz to 150 MHz gives 1.19 µs per slice. Changing only the links from 400 to 25 Gbit/s gives 1.96 µs. Either change alone gives an interval longer than a 1.1 µs round. The 64 devices in the faster case serve a single code patch. The model does not estimate the purchase cost, power or control hardware for such a system.

No single-board 3D allocation in this model passes the distance-33 resource screen. This applies to the storage and port arrangements searched here, not to every possible implementation. The reused-arithmetic 3D core in Chapter 6 does fit distance 49 on a larger FPGA, but takes much longer (Section 6.7). Capacity and speed are separate limits.

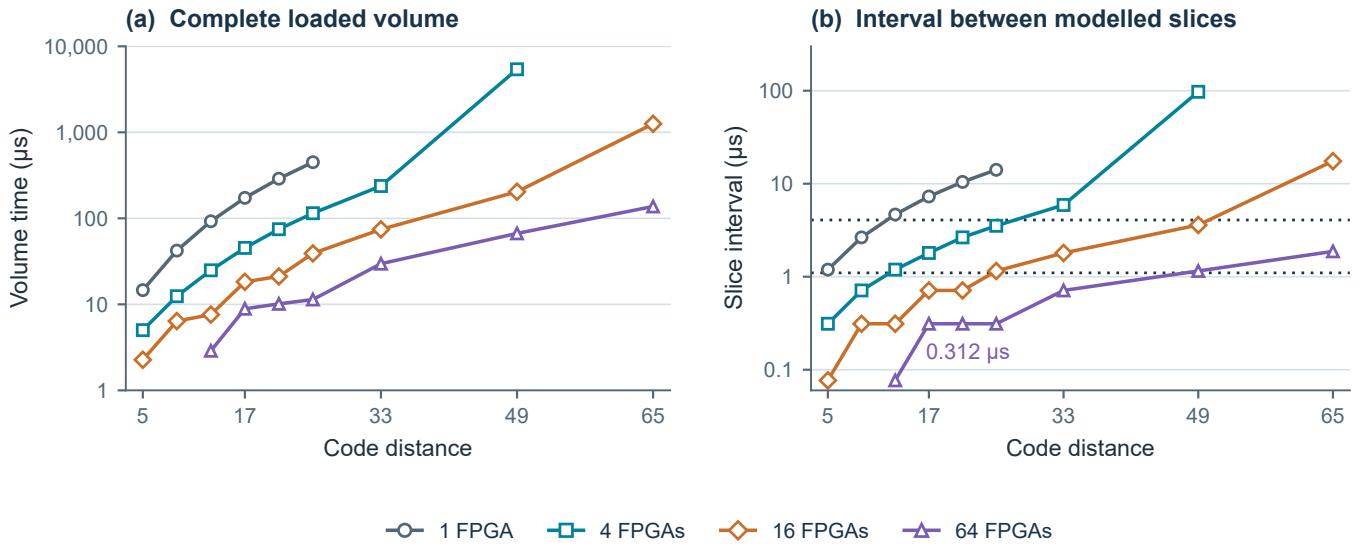


Figure 8.9: Model estimates for 3D volumes and slices

(a) Time to finish a loaded volume with $R = d$. (b) Interval between slices in the same allocation. On 64 devices, the largest tile is 2×2 at distances 17, 21 and 25, giving 0.312 µs per slice. At distance 33 it grows to 3×3. Dotted lines mark the measured Willow round and the round increment compiled for the IBM quantum processor. Curves end where no tested allocation fits. The model assumes VP1902 devices at 250 MHz and 400 Gbit/s per link direction.

8.10 Plateaus and steps in the scaling curves

A local CNN offers a simple ideal case. Its layer count and the area seen by each output are fixed. A surface-code slice has $O(d^2)$ sites. If the number of workers grows in proportion to that area, each worker can keep a tile of fixed size. With inputs already distributed, fixed clock speed, enough local memory ports and constant-bandwidth links to adjacent tiles operating in parallel, the local CNN work per slice can then take $O(1)$ time. This argument concerns the local prediction step. Delivering all the inputs and collecting the outputs requires a total data rate that grows with the number of sites. Global cleanup must still resolve chains that can span the patch.

Figures 8.8 and 8.9 evaluate a fixed hardware budget. Along each curve the board count stays fixed as distance increases. At each distance the model selects the fastest allocation that fits, so adjacent points can use different numbers of arithmetic lanes or memory banks.

In 2D, very small tiles can be inefficient. A five-site halo can reach across several workers, and the longest route on the 64-board mesh drops from six hops at distance 9 to two at distance 21 as the tiles grow. At first, the shorter routes save more time than the larger tiles add in computation. From distance 21 to 65, the largest tile grows from 6×6 to 17×17, but the model can use more arithmetic in parallel. Predicted frame time rises only from 0.89 to 1.13 µs.

By distance 97, the same 64 boards need 2.37 µs. A comparison of distances 65 and 97 shows the source of the additional time. Local processing grows from 0.48 to 1.42 µs, while halo exchange, including reading and writing boundary data, grows from 0.65 to 0.95 µs. Both costs rise. Local computation accounts for about three quarters of the increase in this part of the curve. At distance 33 the balance is different, with about 0.39 µs of processing and 0.58 µs of exchange. Faster arithmetic alone would leave that exchange cost.

The 3D curve has visible steps. At distances 17, 21 and 25, an input plane is ten, twelve or fourteen cells wide. When each plane is split across eight boards in each direction, the busiest board still receives a 2×2 tile in all three cases. The model selects the same arithmetic and predicts the same interval of 0.312 µs per slice. At distance 33 the largest tile grows to 3×3 and the interval rises to 0.712 µs. A complete volume also contains more slices as distance increases. Even across the flat part of the slice curve, the time for the whole volume grows from 8.88 to 11.38 µs.

Each curve stops at the last evaluated distance for which an allocation fits. Predicting the next step would require checking the larger tile's memory and link requirements.

8.11 History buffering for a continuous stream

Rounds arriving during decoding must remain buffered until the decoder can process them. In the four-device distance-17 model, slices arrive every 1.1 µs and take 1.8 µs each to process. With unlimited storage, 96 input planes accumulate over 256 slices. Limiting the main buffer to eighteen planes and the skip path to 64 planes keeps stored inputs intact, but arrivals must wait for space. The longest wait reaches 134.452 µs in the model. A longer run at the same rates would continue to accumulate incoming work. A live interface needs enough throughput to match the arrival rate and enough storage for short bursts.

A rolling implementation would need to handle three phases. At the start of a shot, the buffer is filling and the earliest rounds use the preparation boundary specified by the circuit. During continued operation, each new round extends the available history. An old slice can be released only after every calculation that needs it has finished, including the skip connection. Reusing its memory too early would make a later calculation read the next round's data under the old address. At the end of the shot, final data-qubit measurements supply a different boundary, and the remaining predictions must be completed with that rule. These transitions explain why turning a finite-volume core into a streaming decoder requires more than repeatedly asserting its start signal. The decoder must keep each slice until its last use and associate each correction with the right round.

8.12 Further integration work

Further integration work requires a continuous test that accepts measurement records, retains the CNN's required history, runs cleanup and returns each correction before the controller needs it. In this thesis those boundaries were tested in stages. The Arty and resident tests began with loaded inputs. The distributed tests included loading and collection. The spatial cleanup test reached the combined correction through distance 33 in RTL. A physical distributed test would also need to cover the links and the other worker geometries. Section 9.3 sets out how a continuous test could count both incorrect and late answers.

9. Conclusion

A central practical question was whether a small FPGA could run a neural decoder accurately and quickly enough to help correct errors in a quantum computer. Board tests timed two networks on the Arty. Decoding experiments measured how their predictions changed the complete decoder's failure rate, and simulations examined larger FPGAs and several FPGAs working together.

9.1 Answers to the research questions

RQ1: numerical correctness. The 2D network uses eight-bit weights, keeps biases in accumulator units and avoids rescaling between layers. The trained coefficients give proven width bounds of 13, 21 and 32 bits. For the 3D network, explicit shifts and rounding define how each layer's accumulated result is converted to the next layer's input. Comparing intermediate values helps locate arithmetic faults that a final-decision check might not detect. Protocol simulation, timing analysis and repeated transactions test different parts of the implementation (Section 4.5). Across seven 2D and two 3D board configurations, all 27,086,000 returned decision bits matched the integer reference. Separately written references and additional 2D RTL checks at distances 5, 9, 17 and 33 support this result.

RQ2: speed and resource limits. For the 2D network the main limit is how quickly memory supplies operands. Banking the feature maps lets several kernel taps be read together, with a full eleven-tap row in the first layer. Combined with starting layers as soon as their inputs are ready, this reduced the distance-5 time from 226.32 µs to 49.38 µs. Additional lanes at a timing-qualified 90 MHz reduced it to 38.73 µs, and distance 33 runs in 1.154 ms (Table 5.1). The tested banked configurations have fixed cycle counts that match the scheduling expression (Section 5.4). For the 3D network the first limit is storage. Using four-bit weights in the two middle layers lets the weights fit in the Arty's block memory. Starting layers when their inputs became ready and processing more hidden channels in parallel then reduced the distance-5 time from 5.569 to 2.354 ms (Table 6.1). The fastest passing builds have little timing margin, limiting further changes on this board.

RQ3: complete-decoder quality. A single CNN pass had its clearest benefit with HDRG. At distance 33 with perfect measurements and a physical error probability of 13%, a floating-point pass reduced the HDRG failure rate from 27.0% to 12.9% (Section 7.1.1). The integer hardware model gave 13.3% on the same trials. For MWPM on the same trials the change was much smaller, from 5.49% to 5.13%. At 10% physical error probability the CNN slightly increased MWPM failures. Correlated matching performed better alone in the 2D study. In the 3D study the CNN helped PCM matching, but increased failures in several comparisons using the circuit's detector error model (Section 7.2). On the final-measurement records from the IBM quantum processor, the CNN often increased matching failures and gave no clear improvement over MWPM (Section 7.3.1). A CNN can remove more detection events with each pass and still leave the global decoder with a harder logical decision (Section 7.2.2). These results apply to the two pretrained networks and the conditions tested here. Other trained CNN pipelines improve matching, as discussed in Section 7.4.

RQ4: connection to a quantum controller. The decoder needs to know which check produced each measurement and in which round. The controller then needs the correction before the next operation that depends on it. The centred 3D network already waits four rounds for future measurements, or 16.3 µs on the schedule compiled for the IBM

quantum processor (Section 8.2). On the Arty, processing a distance-5, five-round volume takes about 580 such round intervals. At distance 33 and 1.1 µs per round, even the incoming stream needs about 1.2 Gbit/s with the assumed record headers (Section 8.4).

The timing requirements vary across quantum platforms. The examples in Section 8.3 include trapped-ion adaptive cycles lasting less than 200 ms [47], neutral-atom coherent extraction rounds of 4.45 ms with ancilla readout later [48], and superconducting check rounds of 1.1 µs [8]. These intervals cover different operations. At distance 5, the fastest Arty designs took 38.73 µs for one spatial frame and 2.354 ms for a five-round history. Both are shorter than the 4.45 ms extraction example but much longer than Willow's 1.1 µs round. On each platform, the time available to the decoder runs from the moment the classified measurements are ready to the first operation that needs the correction. Network accuracy must be tested on that platform's records, and the time for data transfer and global cleanup must also be included.

The CNN shortened the measured CPU cleanup calls by factors of 1.3 to 4.3 (Table 8.1). The spatial CNN and HDRG were also tested together in FPGA simulation. On 32 distance-33 frames at 13% physical error probability, the CNN shortened the HDRG search on every frame. Reading the CNN output, searching and combining the corrections added a median of 32.79 µs at the routed 100 MHz clock. Including inference on this Arty-style core, HDRG alone finished sooner on every trial.

RQ5: scaling through overlap and partitioning. Starting a layer as soon as its inputs are ready lets several layers work concurrently (Section 8.7). A three-lane FPGA worker using row-ready scheduling completes distance-33 2D inference in 2,180 simulated cycles and meets internal timing at 100 MHz on the VP1902, corresponding to 21.8 µs. The Arty core took 53 times as long. This worker streams predictions as it writes them. The connected circuit then used fewer cycles than HDRG alone on all 64 frames drawn at the 13% error rate. On the frames tested at 1% and 5%, HDRG alone was faster, because the CNN takes 2,180 cycles regardless of the input (Table 8.2). The VP1902 design uses much more logic, and the complete CNN–HDRG circuit still needs routing and timing checks before measurement on a physical board.

In the multi-FPGA simulation, each FPGA worker computed one tile and used neighbouring inputs at its edges. Combining the tiles reproduced the single-device predictions. The time saved depended on the bandwidth between workers. With a fixed total budget of 32 Gbit/s, four workers were fastest through distance 49, while additional workers increased communication costs (Section 8.8). An architecture model suggests that 64 large FPGAs could provide a 0.71 µs interval between processed slices at distance 33, below the 1.1 µs round period. That conclusion depends on a 250 MHz clock and 400 Gbit/s links that have not been demonstrated (Section 8.9).

In the model, increasing distance sometimes leaves the largest tile unchanged. The time curve can therefore stay flat for several distances before that tile grows again. In the RTL tests, faster arithmetic eventually left workers waiting for memory transfers or for the switch to carry their results. The flat intervals in the model cover only the distances for which the tiles and their data still fit (Section 8.10).

9.2 Limitations

- **Continuous operation.** The designs process finite frames. A live decoder still needs continuous input, a rolling history, cleanup and a controller interface, as Section 8.12 explains.
- **Hardware measurements.** The CNN cores were measured on the Arty. Two ways to connect the CNN to HDRG were simulated. The banked design also passed timing after routing. The streamed connection has only been tested in simulation. The larger-device and multi-worker studies use routed components, simulated links and architecture estimates. A physical board-to-board test is still needed.
- **Timing margins.** The two readiness-based 3D Arty builds meet timing with margins of only a few picoseconds (Table 6.1). They passed the recorded tests, but any change to the design, tools or device needs a fresh timing check.
- **Noise models and data.** The 2D simulations use independent depolarising errors and perfect checks. The main 3D comparison uses one circuit-noise model. Section 7.2.3 tests a lower-noise variant and shows how the ranking changes. The hardware records come from a single IBM quantum processor, so drift and dependence between shots are not covered by the intervals.
- **Inherited networks.** Neither network was trained in this project, and the original training data and history were not available for an overlap check against evaluation inputs. The 2D network was trained for a different noise model from the IBM quantum processor, which may explain part of its weaker performance there. Whether retraining on device-like noise would change the conclusions remains an open question.
- **Statistics.** Paired 95% intervals are reported for each comparison separately, with no correction for the many comparisons in the thesis. Rates estimated from only a few failures remain uncertain, and when an interval spans zero, either decoder could have the lower failure rate.

9.3 Future work

In the connected distance-3 test, reading the CNN predictions accounted for the largest component of latency after inference (Appendix B.8). The streamed distance-33 design shows how this cost can be reduced (Section 8.7). Replaying the full measurement path would identify the main remaining delays before connection to a quantum controller.

1. **Replay measurements at the controller rate.** Build the detection-event front end and a rolling window for the 3D network, and feed it measurements recorded on the IBM quantum processor at the rate specified by the circuit schedule, with stalls, missing records and resets. Add cleanup, at first on a host, and time the complete path from the last measurement to the returned correction.
2. **Compare decoders within the same time budget.** Include fast cleanup decoders such as union-find [31], and retrain or fine-tune the network on device-like noise. The hardware choice should follow the performance of the complete decoder.
3. **Build the large single-device design.** Route the resident worker together with the HDRG engines. Measure the combined decoder and the larger 3D cores on physical boards.

4. **Two physical boards.** Connect two FPGAs with real links, exchange halos and reproduce the single-device result, measuring framing, flow control and collection.
5. **A live experiment.** With a partner who can provide access to a quantum controller's measurement and feedback interfaces, run a memory experiment in which the decoder's corrections arrive before the operation that needs them.

The replay test would match each returned correction to the offline answer for the same shot and round. Timestamps at input arrival, completion of inference, completion of cleanup and delivery to the controller would separate computation from transfer delays. Communication and control can be a substantial part of the total response. In their nine-round experiment, Caune et al. measured 3.1 µs for these stages within a 9.6 µs response to a conditional gate [13] (Section 8.3). Running the replay test continuously would show whether the queue stays bounded and how often a correction misses its deadline. The same check should be repeated across the physical link before the decoder guides an operation on a quantum processor.

9.4 Closing remarks

This thesis implemented two convolutional neural-network decoders for the surface code on FPGAs and evaluated them as parts of a complete decoder. On the Arty A7-100T, every tested decision matched the integer reference. At distance 5, inference time fell from 226.32 µs to 38.73 µs for the 2D network and from 5.569 ms to 2.354 ms for the 3D network. The same 2D core processed a distance-33 frame in 1.154 ms. These gains required memory organisation and scheduling that supplied the arithmetic units efficiently. Memory read bandwidth limited the 2D design, while weight storage was the first constraint on the 3D design. The number of arithmetic units alone is therefore a poor measure of decoder performance.

The decoding experiments showed that the benefit of the CNN depends on the global decoder that follows it. At distance 33 in the perfect-check simulation, one pass approximately halved the HDRG failure rate and gave a smaller gain with MWPM. It increased failures with correlated matching and in several circuit-noise comparisons using DEM matching. These findings apply to the two pretrained networks and the conditions tested. They support training and evaluating a network together with its intended cleanup decoder.

Quantum error correction needs a decoder on every platform, but the time available differs. Published trapped-ion and neutral-atom experiments report millisecond-scale rounds (Section 8.3). The improved Arty inference times are shorter than the 4.45 ms neutral-atom extraction round. These measurements support integration tests on slower platforms, where there can be more time for both inference and cleanup. A live test must establish the full measurement-to-correction time and the logical failure rate for the chosen processor.

Superconducting processors produce measurements much more frequently, with check rounds of 1.1 µs on Willow. The larger single-FPGA worker completed distance-33 inference in 2,180 simulated cycles, equivalent to 21.8 µs at its routed 100 MHz clock. In connected simulations, the faster CNN and HDRG used fewer cycles than HDRG alone at the higher sampled error rates, but not at the lower rates.

Multi-worker RTL simulations reproduced the undivided predictions through distance 65 with up to 64 workers. The fastest tested 64-worker configuration completed a distance-33 frame in 50.6 µs, including loading, exchange and collection but excluding global cleanup (Table 8.3). A separate model of 64 VP1902 devices projects a preloaded distance-33 spatial frame in 0.969 µs and a 33-round history volume in 29.7 µs, assuming 250 MHz clocks and 400 Gbit/s links (Section 8.9). These projections exclude global cleanup and still require physical verification. The projected interval between history slices is 0.712 µs, below Willow's 1.1 µs round period. If a linked-FPGA implementation sustained that interval, CNN inference would keep pace with Willow's 1.1 µs check rounds.

References

[1] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018, doi: 10.22331/q-2018-08-06-79.

[2] J. J. Goings *et al.*, "Reliably assessing the electronic structure of cytochrome P450 on today's classical computers and tomorrow's quantum computers," *Proceedings of the National Academy of Sciences*, vol. 119, no. 38, p. e2203533119, 2022, doi: 10.1073/pnas.2203533119.

[3] N. Stamatopoulos *et al.*, "Option pricing using quantum computers," *Quantum*, vol. 4, p. 291, 2020, doi: 10.22331/q-2020-07-06-291.

[4] V. Havlíček *et al.*, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, 2019, doi: 10.1038/s41586-019-0980-2.

[5] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, p. 032324, 2012, doi: 10.1103/PhysRevA.86.032324.

[6] B. M. Terhal, "Quantum error correction for quantum memories," *Reviews of Modern Physics*, vol. 87, no. 2, pp. 307–346, 2015, doi: 10.1103/revmodphys.87.307.

[7] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002, doi: 10.1063/1.1499754.

[8] Google Quantum AI and Collaborators, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, no. 8052, pp. 920–926, 2025, doi: 10.1038/s41586-024-08449-y.

[9] O. Higgott and C. Gidney, "Sparse blossom: Correcting a million errors per core second with minimum-weight matching," *Quantum*, vol. 9, p. 1600, 2025, doi: 10.22331/q-2025-01-20-1600.

[10] S. Gicev, L. C. L. Hollenberg, and M. Usman, "A scalable and fast artificial neural network syndrome decoder for surface codes," *Quantum*, vol. 7, p. 1058, 2023, doi: 10.22331/q-2023-07-12-1058.

[11] S. Gicev, L. C. L. Hollenberg, and M. Usman, "Fully convolutional 3D neural network decoders for surface codes with syndrome circuit noise." 2025. Available: https://arxiv.org/abs/2506.16113v1

[12] L. Stefanazzi *et al.*, "The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors," *Review of Scientific Instruments*, vol. 93, no. 4, p. 044709, 2022, doi: 10.1063/5.0076249.

[13] L. Caune *et al.*, "Demonstrating real-time and low-latency quantum error correction with superconducting qubits," *Nature Communications*, vol. 17, no. 1, p. 7383, 2026, doi: 10.1038/s41467-026-73331-6.

[14] Digilent, "Arty A7-100T: Artix-7 FPGA development board." Official product specifications. Available: https://digilent.com/shop/arty-a7-100t-artix-7-fpga-development-board/

[15] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021, doi: 10.22331/q-2021-04-15-433.

[16] C. Gidney, "How to factor 2048 bit RSA integers with less than a million noisy qubits." 2025. Available: https://arxiv.org/abs/2505.15917

[17] C. Chamberland, L. Goncalves, P. Sivarajah, E. Peterson, and S. Grimberg, "Techniques for combining fast local decoders with global decoders under circuit-level noise," *Quantum Science and Technology*, vol. 8, no. 4, p. 045011, 2023, doi: 10.1088/2058-9565/ace64d.

[18] J. Preskill, "Lecture notes for Ph219/CS219: Quantum information, chapter 2: Foundations i: States and ensembles." California Institute of Technology course notes, Jul. 2015. Available: https://www.preskill.caltech.edu/ph219/chap2_15.pdf

[19] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, pp. 802–803, 1982, doi: 10.1038/299802a0.

[20] Y. Tomita and K. M. Svore, "Low-distance surface codes under realistic quantum noise," *Physical Review A*, vol. 90, no. 6, p. 062320, 2014, doi: 10.1103/PhysRevA.90.062320.

[21] A. G. Fowler, "Optimal complexity correction of correlated errors in the surface code." 2013. Available: https://arxiv.org/abs/1310.0863v1

[22] PyMatching contributors, *PyMatching 2: Correlated matching*. Available: https://github.com/oscarhiggott/PyMatching

[23] AMD, *7 series FPGAs configurable logic block user guide (UG474)*, 1.9. Advanced Micro Devices, 2025. Available: https://docs.amd.com/r/en-US/ug474_7Series_CLB/CLB-Slices

[24] Xilinx, *7 series FPGAs memory resources user guide (UG473)*, 1.14. Xilinx, 2019. Available: https://docs.amd.com/v/u/en-US/ug473_7Series_Memory_Resources

[25] Xilinx, *7 series DSP48E1 slice user guide (UG479)*, 1.10. Xilinx, 2018. Available: https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1

[26] Xilinx, *7 series FPGAs data sheet: Overview (DS180)*, 2.6.1. Xilinx, 2020. Available: https://docs.amd.com/v/u/en-US/ds180_7Series_Overview

[27] AMD, *Vitis high-level synthesis user guide (UG1399): Pipelining paradigm*, 2026.1. Advanced Micro Devices, 2026. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls/Pipelining-Paradigm

[28] S. Maurya and S. Tannu, "Managing classical processing requirements for quantum error correction." 2024. Available: https://arxiv.org/abs/2406.17995v1

[29] B. Barber *et al.*, "A real-time, scalable, fast and resource-efficient decoder for a quantum computer," *Nature Electronics*, vol. 8, pp. 84–91, 2025, doi: 10.1038/s41928-024-01319-5.

[30] O. Higgott, "PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, 2022, doi: 10.1145/3505637.

[31] N. Delfosse and N. H. Nickerson, "Almost-linear time decoding algorithm for topological codes," *Quantum*, vol. 5, p. 595, 2021, doi: 10.22331/q-2021-12-02-595.

[32] G. Torlai and R. G. Melko, "Neural decoder for topological codes," *Physical Review Letters*, vol. 119, p. 030501, 2017, doi: 10.1103/PhysRevLett.119.030501.

[33] S. Varsamopoulos, B. Criger, and K. Bertels, "Decoding small surface codes with feedforward neural networks," *Quantum Science and Technology*, vol. 3, p. 015004, 2018, doi: 10.1088/2058-9565/aa955a.

[34] C. Chamberland and P. Ronagh, "Deep neural decoders for near term fault-tolerant experiments," *Quantum Science and Technology*, vol. 3, p. 044002, 2018, doi: 10.1088/2058-9565/aad1f7.

[35] S. Gicev, L. C. L. Hollenberg, and M. Usman, "Fully convolutional 3D neural network decoders for surface codes with syndrome circuit noise," *Quantum Science and Technology*, vol. 11, no. 2, p. 025048, 2026, doi: 10.1088/2058-9565/ae5fc9.

[36] S. Gicev, L. C. L. Hollenberg, and M. Usman, "Quantum computer error structure probed by quantum error correction syndrome measurements," *Physical Review Research*, vol. 6, p. 043249, 2024, doi: 10.1103/PhysRevResearch.6.043249.

[37] K. Meinerz, C.-Y. Park, and S. Trebst, "Scalable neural decoder for topological surface codes," *Physical Review Letters*, vol. 128, no. 8, p. 080505, 2022, doi: 10.1103/PhysRevLett.128.080505.

[38] J. Bausch *et al.*, "Learning high-accuracy error decoding for quantum processors," *Nature*, vol. 635, no. 8040, pp. 834–840, 2024, doi: 10.1038/s41586-024-08148-8.

[39] R. W. J. Overwater, M. Babaie, and F. Sebastiano, "Neural-network decoders for quantum error correction using surface codes: A space exploration of the hardware cost-performance tradeoffs," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–19, 2022, doi: 10.1109/TQE.2022.3174017.

[40] M. Zhang *et al.*, "A scalable, fast and programmable neural decoder for fault-tolerant quantum computation using surface codes." 2023. Available: https://arxiv.org/abs/2305.15767

[41] G. Yan, S. Li, and Y. Du, "Rethink the role of neural decoders in quantum error correction," in *Proceedings of the 43rd international conference on machine learning*, in Proceedings of machine learning research, vol. 306. PMLR, 2026, pp. 143726–143759. Available: https://proceedings.mlr.press/v306/yan26g.html

[42] C. Chamberland, J. Olle, M. Li, S. Thornton, and I. Baratta, "Fast and accurate AI-based pre-decoders for surface codes." 2026. Available: https://arxiv.org/abs/2604.12841v1

[43] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2018. doi: 10.1109/CVPR.2018.00286.

[44] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, ACM, 2017, pp. 65–74. doi: 10.1145/3020078.3021744.

[45] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, vol. 13, p. P07027, 2018, doi: 10.1088/1748-0221/13/07/P07027.

[46] X. Yang *et al.*, "Real-time surface-code error correction using an FPGA-based neural-network decoder." 2026. Available: https://arxiv.org/abs/2605.04892v1

[47] C. Ryan-Anderson *et al.*, "Realization of real-time fault-tolerant quantum error correction," *Physical Review X*, vol. 11, no. 4, p. 041058, 2021, doi: 10.1103/PhysRevX.11.041058.

[48] D. Bluvstein *et al.*, "A fault-tolerant neutral-atom architecture for universal quantum computation," *Nature*, vol. 649, no. 8095, pp. 39–46, 2026, doi: 10.1038/s41586-025-09848-5.

[49] Atom Computing and Collaborators, "Quantum error correction with the toric code." 2026. Available: https://arxiv.org/abs/2606.04079v1

[50] L. Skoric, D. E. Browne, K. M. Barnes, N. I. Gillespie, and E. T. Campbell, "Parallel window decoding enables scalable fault tolerant quantum computation," *Nature Communications*, vol. 14, no. 1, p. 7040, 2023, doi: 10.1038/s41467-023-42482-1.

[51] X. Tan, F. Zhang, R. Chao, Y. Shi, and J. Chen, “Scalable surface-code decoders with parallelization in time,” *PRX Quantum*, vol. 4, no. 4, p. 040344, 2023, doi: 10.1103/PRXQuantum.4.040344.

[52] Y. Wu and L. Zhong, “Fusion blossom: Fast MWPM decoders for QEC,” in *2023 IEEE international conference on quantum computing and engineering (QCE)*, 2023, pp. 928–938. doi: 10.1109/qce57702.2023.00107.

[53] S. Varsamopoulos, K. Bertels, and C. G. Almudever, “Decoding surface code with a distributed neural network-based decoder,” *Quantum Machine Intelligence*, vol. 2, p. 3, 2020, doi: 10.1007/s42484-020-00015-9.

[54] M. Garg, “FPGA-accelerated surface-code ANN decoding for real-time QEC,” Unpublished MCS research proposal, University of Melbourne.

[55] E. B. Wilson, “Probable inference, the law of succession, and statistical inference,” *Journal of the American Statistical Association*, vol. 22, no. 158, pp. 209–212, 1927, doi: 10.1080/01621459.1927.10502953.

[56] T. Tango, “Equivalence test and confidence interval for the difference in proportions for the paired-sample design,” *Statistics in Medicine*, vol. 17, no. 8, pp. 891–908, 1998, doi: 10.1002/(SICI)1097-0258(19980430)17:8<891::AID-SIM780>3.0.CO;2-B.

[57] M. W. Fagerland, S. Lydersen, and P. Laake, “Recommended tests and confidence intervals for paired binomial proportions,” *Statistics in Medicine*, vol. 33, no. 16, pp. 2850–2875, 2014, doi: 10.1002/sim.6148.

[58] C. Gidney, “Stim: A fast stabilizer circuit simulator,” *Quantum*, vol. 5, p. 497, 2021, doi: 10.22331/q-2021-07-06-497.

[59] AMD, *UltraScale architecture and product data sheet: Overview (DS890)*, 4.10. AMD, 2026. Available: https://docs.amd.com/v/u/en-US/ds890-ultrascale-overview

[60] AMD, *Versal architecture and product data sheet: Overview (DS950)*, 2.11. AMD, 2026. Available: https://docs.amd.com/v/u/en-US/ds950-versal-overview

A. Numerical details

Each register must be wide enough to hold the largest value the network could produce. This appendix gives the numerical bounds used in Section 4.4. Those limits were calculated from the weights used in the FPGA. Changing the weights would require checking the limits again.

A.1 Derivation of the bounds

Each output is a bias b plus a sum of products. An input x_i can lie between zero and U_i. To find the lowest possible sum, use the largest allowed input for every negative weight and zero for every positive weight. Reversing that choice gives the upper limit:

$$a_{\min} = b + \sum_{w_i<0} w_i U_i, \qquad a_{\max} = b + \sum_{w_i>0} w_i U_i.$$

At the input, each bit is either zero or one, so $U_i = 1$. In a hidden layer, ReLU replaces negative results with zero. If the upper limit for an output is positive, it also bounds the value sent to the next layer. If it is zero or negative, ReLU always sends zero. A signed B-bit register can hold any integer from -2^{B-1} to $2^{B-1} - 1$.

The FPGA adds some products together before others. A partial sum can exceed the final sum before a negative term is added. The running total after each group of products was therefore checked, as well as the final sum, for all tested groups of one to eleven taps.

A.2 2D network

Table A.1: Bounds of the 2D integer network for binary inputs. The envelopes over all partial sums and all lane groupings also fit in 13, 21 and 32 bits.

Layer	Range of the final sum	Bits needed	Accumulator in hardware	Stored activation
1	−3,843 to 638	13	16	11 bits
2	−987,422 to 543,106	21	24	21 bits
3	−2,120,775,324 to 909,729,481	32	36	decision bit

The largest hidden values are 638 and 543,106. They fit in the 11-bit and 21-bit signed registers used to store them. At the output, a sigmoid exceeds one half exactly when its input z is positive. The FPGA can therefore return one for $z > 0$ and zero otherwise. Testing only the sign bit would mishandle zero. This rule gives the same decision for a given z, although rounding a weight can change z itself.

The rule was checked on 500 generated frames. All 4,225,000 decisions agreed. A separate calculation that read the coefficients from the hardware files also reproduced every bound in Table A.1.

Figure A.1 compares the required bits with the bits used in hardware. It also shows the largest values reached on four test inputs. Those measurements illustrate how much of the range these inputs used. The bounds cover every binary input for these fixed coefficients.

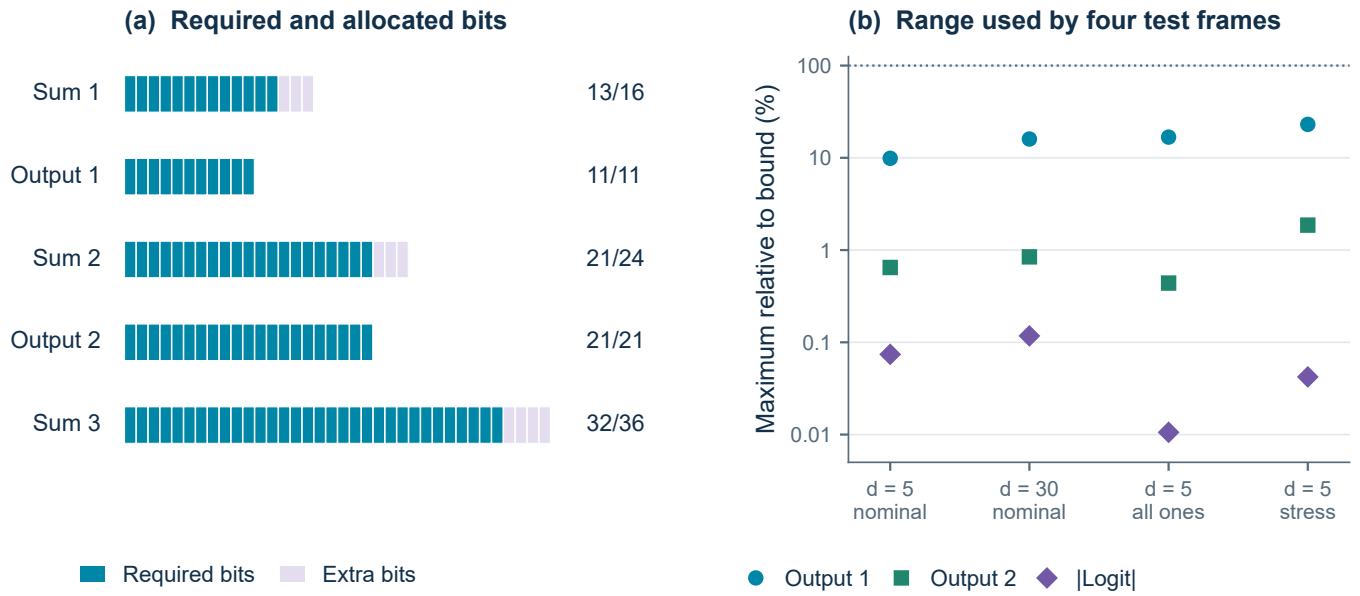


Figure A.1: Required and allocated bit widths in the 2D datapath

(a) Filled cells show the bits needed for each layer's signed accumulator and each stored activation. Pale cells show the extra bits allocated in hardware. The bounds hold for any binary input and the tested lane groupings. (b) Points show the largest activation or absolute logit on each of four test frames as a percentage of its bound. The vertical scale is logarithmic.

A.3 3D network

Each of the three hidden layers scales its result before passing it on. The scale uses a multiplier for each channel and a right shift for each layer. The shifts are 15, 20 and 19 bits. Table A.2 gives the largest possible accumulated sum and the largest product after adding the rounding term from Section 4.3.

Table A.2: Bounds of the 3D integer network with binary inputs and eight-bit hidden activations.

Hidden layer	Bound on $\lvert A \rvert$	Largest $\max(A, 0)M + 2^{S-1}$
1	18,909	91,173,184
2	3,152,043	31,726,251,230
3	1,126,655	11,260,581,184

Each non-negative hidden sum is below 2^{25}, so the circuit can pass its lowest 25 bits to the multiplier without losing information. The product, including rounding, stays below 2^{47} and fits in the signed 48-bit DSP path. The final output also includes the skip connection. Its absolute value is at most 4,310,974, which fits in a signed 32-bit register. These limits were checked in NumPy, and the output of every layer was compared with the reference.

B. Implementation details

The board tests time the CNN from its start signal to the last prediction. The distributed tests start with the first input transfer and include the time spent moving data between workers. The final section follows a CNN prediction through global cleanup to the combined correction. Chapters 5 and 6 discuss the board results, while Sections 8.7 and 8.8 compare the distributed designs.

B.1 The board interface

The host sends a complete frame over the 115,200-baud serial link (Figure B.1). The board starts the CNN, waits for it to finish and returns the cycle count and the decisions. For the distance-5 3D test, the host sends 96 positions, each with twelve input bits packed into two bytes. That makes 192 input bytes. The reply contains 144 bytes of decisions. Including two command bytes in the request, and framing plus a four-byte counter in the reply, the complete transfers contain 194 and 150 bytes respectively. The board keeps a second copy of the input for the skip connection.

The counter starts when the CNN accepts the command and stops when it writes its last decision. It excludes the serial transfer. The 3D wrapper first waits for the core to signal that it is busy. Without that step, the wrapper could read the previous frame's completion signal before the new frame had started (Section 6.6).

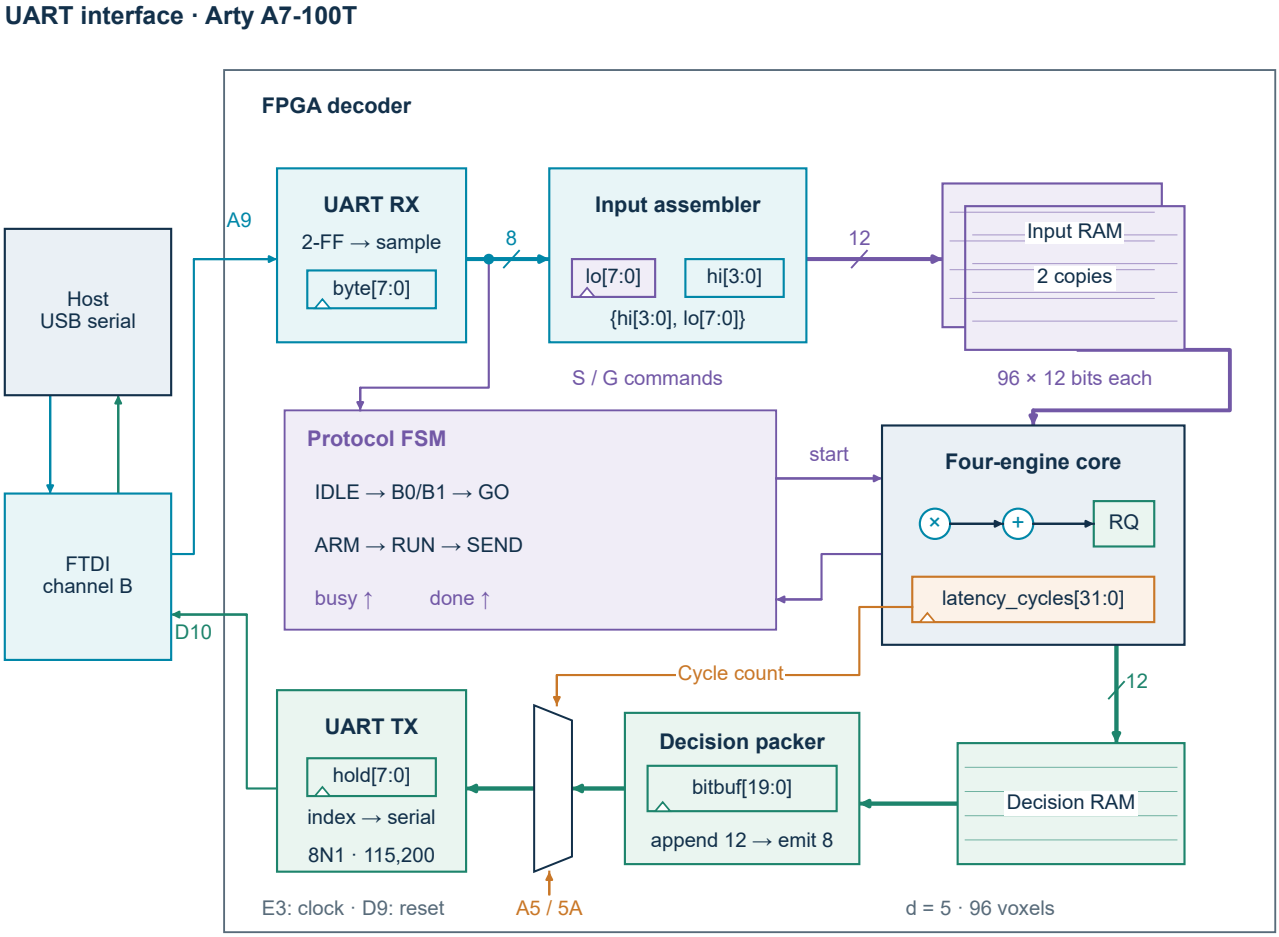


Figure B.1: The board test interface

A USB-UART link at 115,200 baud carries a complete input frame to the board. The illustrated 3D wrapper starts the core, waits for its busy acknowledgement and completion, and returns the cycle counter and packed decisions. At distance 5 this is 192 input bytes and 144 decision bytes. Core timing excludes the serial transfer.

B.2 Layer scheduling checks

Figure B.2 shows when each 2D layer starts and stops issuing calculations under the schedules discussed in Section 5.4. The final result reaches memory a few cycles after the last calculation starts. With banked memory, each layer receives its inputs sooner and the layers can overlap. At distance 5 the banked simulation finished in 4,938 cycles. The shared design took 22,630 cycles in simulation and 22,632 on the board. The retained records do not identify the cause, so the simulation and board counts are reported separately. Both simulations returned the expected 162 decisions.

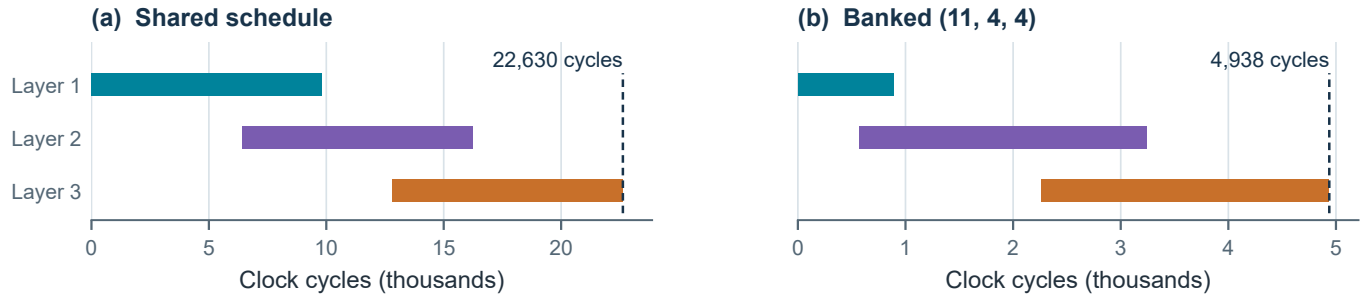


Figure B.2: Issue schedule of the 2D layers

Each bar spans the first through last issue cycle in a distance-5 RTL trace. The two panels use different time scales. The dashed marker includes the final drain, at 22,630 cycles for the shared schedule (22,632 on the board) and 4,938 cycles for the banked design. Both runs matched all 162 expected decision bits.

B.3 Arty timing and capacity limits

Two constraints limited the larger 3D designs on the Arty family. Some needed more logic than the device provides. Others fitted, but their longest paths exceeded the clock budget. With 32 hidden output channels processed together ($TCO_H = 32$), the fixed-slot design of Section 6.5 failed its 100 MHz timing target by 0.324 ns. At 64 channels it exceeded the XC7A100T's logic capacity. At 128 on the larger XC7A200T, Vivado could not complete placement.

Increasing the distance from 5 to 7 made the feature maps exceed the depth available in one group of small memories. The number of LUTs used for storage doubled, from 8,352 to 16,704. Total block-memory use stayed at 133.5 BRAM36 equivalents. This build failed its 100 MHz timing target by 2.137 ns. The 2D synthesis tests also covered hidden layers with one to eleven lanes. Six lanes can process an eleven-tap row in two groups. Eight lanes still need two groups, so the extra hardware does not reduce the cycle count.

A faster XC7A200T speed grade was tested for the distance-30 2D design with $(11, 4, 4)$ lanes. The hardware fitted, but failed timing at its 5 ns clock-period target by 1.702 ns. This run did not establish a usable clock frequency.

B.4 Additional classifier check

Weights trained with simplified correction labels were also tested. They use the same 3D architecture, but the hidden layers shift by 15, 19 and 18 bits when converting a sum to the next layer's input. These weights were checked in a simulation of the board's serial interface. On the distance-5 frame all 1,152 decisions matched their reference, and the CNN took 556,879 cycles. The physical board tests and decoder comparisons used the original weights, with shifts of 15, 20 and 19.

B.5 Components implemented on larger FPGAs

Tables B.1 and B.2 give the Vivado 2026.1 results for five combinations of design and device. The distance-33 resident worker computes three spatial positions in parallel (Section 8.7). The distributed design is worker 5, an interior tile of a sixteen-worker partition, including its queues (Section 8.8). The two 3D cores process 64 hidden output channels together, at distances 5 and 49 (Section 6.7). The resource counts and timing margins come from the design tools.

Table B.1: Resources of the larger-device components. The parts are xcvu13p-flga2577-3-e, xcvp1902-vsvb6865-3HP-e-S and xcvp1802-lsvc4072-3HP-e-S.

Component and device	Stage	LUT	FF	BRAM36	URAM	DSP
2D resident worker, VU13P	Synthesis (routing failed)	914,866	1,234,835	78	0	0
2D resident worker, VP1902	Routed	831,868	1,119,681	102	0	0
2D distributed tile, VU13P	Routed	327,752	435,683	16.5	0	0
3D core, $d = 5$, VU13P	Routed	113,663	36,742	97.5	0	3
3D core, $d = 49$, VP1802	Routed	119,136	34,603	97	780	13

Vivado could not route the resident worker on the VU13P, even after its address calculation was simplified. For the distance-49 3D core on the VP1802, UltraRAM holds a complete volume. A live decoder would also need to manage incoming measurements as older ones leave the buffer.

Table B.2: Internal timing after routing. Setup and hold give the margins at the checked clock. Pulse-width checks also passed, with margins of 4.427 to 7.505 ns. The distance-49 clock period was 12 ns.

Component	Target	Setup at target	Checked clock	Setup	Hold
2D resident worker, VP1902	250 MHz	−4.587 ns	100 MHz	+1.415 ns	+0.010 ns
2D tile, worker clock	100 MHz	−3.318 ns	62.5 MHz	+2.682 ns	+0.010 ns
2D tile, switch clock	100 MHz	met	100 MHz	+5.024 ns	+0.045 ns
3D core, $d = 5$, VU13P	250 MHz	−5.830 ns	100 MHz	+0.170 ns	+0.011 ns
3D core, $d = 49$, VP1802	250 MHz	−7.092 ns	83.3 MHz	+0.907 ns	+0.012 ns

B.6 Partitioning and the resident worker

The integer calculation was first split in software to check that the tiles gave the same output as the whole map. All 102 cases agreed, including unequal tiles, splits across time, skip connections and up to 64 workers. Section 8.6 explains how the overlapping tile boundaries preserve the whole-map calculation. A separate replay checked 530,017 records passed between tiles. At distance 33 the map is 65×65. Splitting one side into four gives widths of 17, 16, 16 and 16. Each worker reserves space for the largest tile and checks that each incoming record belongs to the region it needs.

The resident-worker tests covered 84 frames across fourteen configurations. These included narrow maps, alternating stored inputs, pauses during reads and resets during a calculation. More than 7.7 million sums, activations and decisions were checked. Eight further distance tests extending to 49, 65 and 97 checked another 21.4 million values. Two additional tests supplied incorrect expected values and attempted to read the output too early. The tests detected both errors.

B.7 The distributed RTL experiment

Each message has 320 bits. Of these, 192 are available for the values at one site. The message also identifies the location and layer. An input site contributes four bits. A site after layer one needs $9 \times 11 = 99$ bits, and one after layer two needs $9 \times 21 = 189$. A final site needs two bits. The message stays the same length in every case. Each worker has a four-message queue in each direction. A queue holds messages while their receiver is busy. When it is full, the sender must wait rather than overwrite a message. Each port can accept one message every 10 ns. The switch can send the same value to several workers, including the worker that produced it. Table B.3 counts these deliveries.

Table B.3: Records per distance-33 frame on the 65×65 map, counting each recipient separately.

Records per frame	4 workers	16 workers	64 workers
Inputs from the host	5,625	9,025	18,225
Intermediate deliveries, all recipients	11,250	18,050	36,450
of which to a different worker	2,800	9,600	28,000
Outputs returned, all three layers	12,675	12,675	12,675
of which final-layer outputs	4,225	4,225	4,225

With 32 Gbit/s shared across the switch, the ports are served in rotation. For 64 workers a given port is served once every 64 cycles, or 640 ns. With 128 Gbit/s shared, the switch can serve four ports per direction in each clock cycle. A port leaves its slot empty if it has nothing to send. Both tests count the entire message, header included. The timer runs from the first accepted input to the last final output. It includes loading, transfers between workers, collection and any memory initialisation still in progress.

The tests cover distances 3, 5, 7, 9, 13, 17, 21, 25, 33, 41, 49 and 65 with one, four, sixteen and 64 workers. Distance 3 has too few outputs to give all 64 workers a site, leaving 47 usable partitions. Six patterns per partition exercised constant, alternating, sparse, dense and single-site inputs. Across the bandwidth, reset, repeat and interface tests, 618 executions matched 47,614,920 sums and the same number of activations. The messages were also replayed in order, to check what each worker received and whether the sender had already produced it. Deliberately changing reference values, messages or traces caused the checks to fail.

At distance 9, queues holding four, eight and sixteen messages were compared. One worker took 72.18, 66.42 and 66.42 µs. Four workers took 43.18, 37.20 and 37.20 µs. In both cases eight entries were enough to reach the best time measured. Increasing both queues from four entries to eight added 2,560 bits per worker, bringing their total to 5,120 bits.

Finally, all sixteen simulated workers used the clock periods that one routed interior tile had met: 16 ns for the worker and 10 ns for the switch. Six ordinary frames took 133.61–133.82 µs. Six fresh frames after a reset during operation took 133.54–133.59 µs. All twelve agreed with the whole-map result. These runs are additional to the 618 above and exclude the interrupted frame. The complete network and the links between devices were simulated.

B.8 Global cleanup on an FPGA

After the CNN makes its prediction, the FPGA computes which checks that prediction explains. The global decoder then finds a correction for the remaining checks. Both the global decoder alone and the complete path from the last CNN prediction to the combined correction were timed. All tests in this section use a 2D patch with perfect check measurements. Clock cycles were counted in RTL simulation and converted with the clock period that each circuit met after routing. Loading the input and sending the correction to a controller are outside these times. Section 8.5 compares this added time with the CNN time and the software cleanup measurements.

Small patches. At distance 3, each component has six check bits, giving 64 possible inputs. MWPM and HDRG were run in software for each one, and their answers were stored on the FPGA. The six bits identify the input and a seventh bit selects the algorithm. The circuit reads out a 13-bit mask, one bit per data qubit (Figure B.3(a)). With both components and both algorithms, the tables need 3,328 bits. Every one of the 8,192 combinations of inputs and modes gave the expected masks. A new input can enter in each clock cycle.

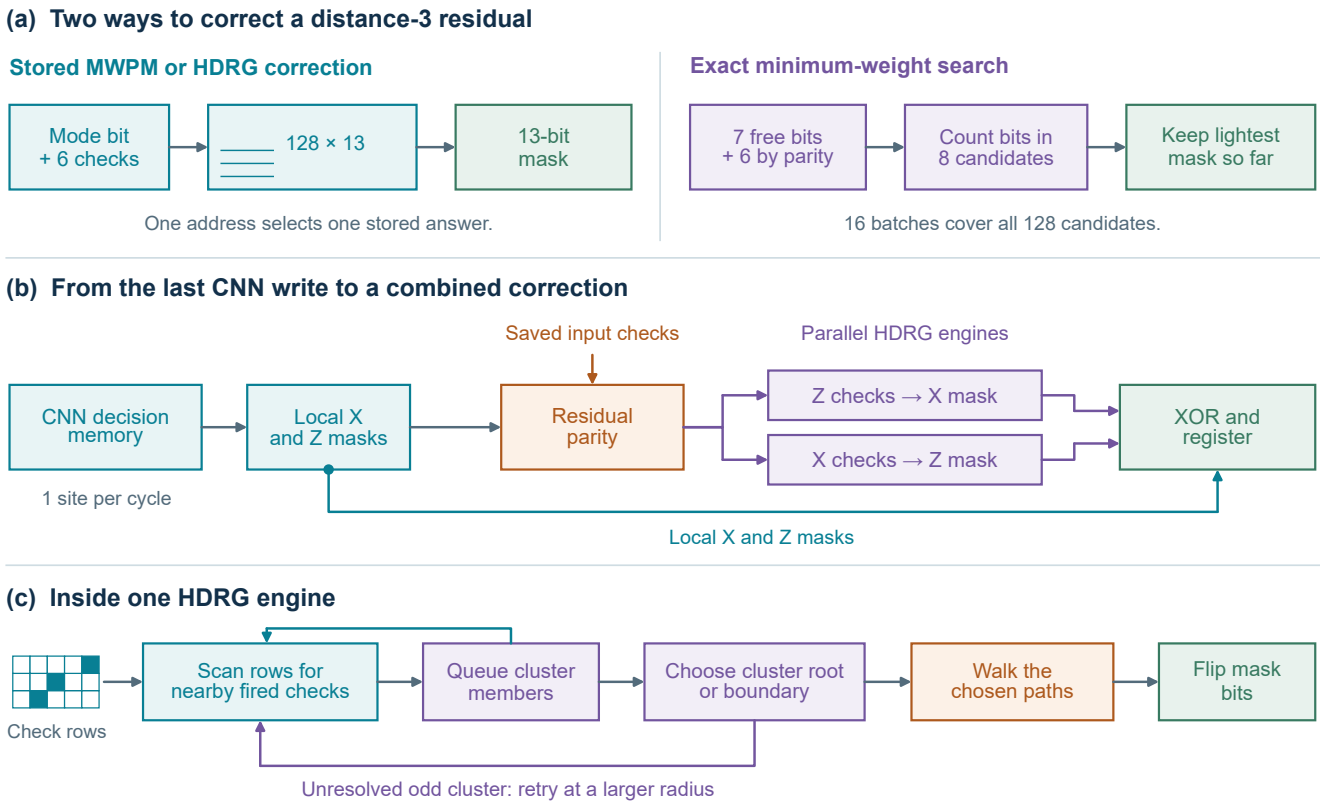


Figure B.3: FPGA circuits for global cleanup after the CNN

(a) For one distance-3 component, the lookup stores the software answers. The minimum-weight circuit searches for the lightest mask consistent with the six checks. (b) The banked CNN stores its predictions before two HDRG engines process the residual. Added latency runs from the last CNN write to the registered combined correction (Table B.5). (c) Each engine queues nearby fired checks and traces paths to clear them. An odd cluster waits for a larger search radius if it cannot reach a boundary.

Table B.4 compares the lookup with two circuits that calculate the answer. The exact minimum-weight circuit evaluates all 128 masks that fit a component's checks and returns one with the fewest flipped bits. It tests eight at a time. All 4,096 input pairs gave minimum-weight corrections. When minimum-weight paths tied, the circuit and PyMatching

returned different masks for 9 of the 64 X inputs and 13 of the 64 Z inputs. Both masks cleared the checks. The choice changed the logical correction for 6 of the 64 X inputs and 9 of the 64 Z inputs.

The small HDRG circuit groups nearby fired checks. A group can be cleared when it has an even number of checks or when it reaches a boundary. Other groups wait while the search expands. This explains the range in the table. Both output masks agreed with software on all 4,096 distance-3 input pairs and 1,235 distance-5 pairs in Vivado simulation. A separate exhaustive test at distance 5 covered all 2^{20} inputs to each component and confirmed that even the slowest pair finished in 27 cycles.

Table B.4: Time from accepting a complete residual to returning both correction masks. All four standalone circuits met 100 MHz after routing on the Arty A7 target and used no DSPs or block RAM.

Core	Distance	Cycles	Response at 100 MHz	LUTs / flip-flops
MWPM or HDRG table	3	1	10 ns	41 / 41
Exact minimum-weight search	3	21	210 ns	282 / 418
HDRG cluster growth	3	1–9	10–90 ns	153 / 85
HDRG cluster growth	5	1–27	10–270 ns	673 / 255

Connecting the lookup to the CNN illustrates the difference between the two timers. Reading 13 predictions and passing them on took 14 cycles. Forming the residual, reading the table and combining the correction took one cycle each. For 32 inputs in both modes the combined masks matched the reference. The lookup answered in 10 ns. The full operation after the last prediction took 170 ns at the routed 100 MHz clock.

Larger patches. The lookup would already need nearly 172 million bits at distance 5. Larger distances use HDRG, with a circuit that reuses the same hardware for each row of checks. Figure B.3(b) follows one frame through the banked CNN and cleanup circuit. In each clock cycle, the circuit reads one site’s X and Z predictions from the CNN memory. For each check, the circuit takes the parity of the neighbouring predictions and XORs it with the saved input bit. A one means that this check still needs a correction. Two engines search the remaining checks in parallel. One builds the X correction mask and the other builds the Z correction mask. When both finish, a final register combines their masks with the CNN predictions.

Figure B.3(c) shows why the search time varies. An engine stores the checks it discovers in a queue. It visits the neighbours of each queued check until the group is complete. To clear the group it follows paths to a common member or a boundary, flipping one mask bit per step. Even an empty patch takes time to scan. The standalone circuit returned the same masks as software on 4,951 inputs at ten distances between 3 and 33. At distance 33 its 95 test inputs ranged from 33 cycles for empty checks to 23,732 for all checks fired. The circuit met an 11.112 ns clock period on the Arty target and 6.667 ns on the Virtex UltraScale+ VU13P.

Table B.5: Clock cycles for the connected CNN and HDRG. The first timing column runs from a loaded frame to the last CNN prediction. Each of the next three gives the additional cycles to the combined correction for the stated input. The noise ranges use four frames per distance and 32 at distance 33. The ten-distance sweep also tested distances 5, 7, 13, 21 and 29.

Distance	CNN cycles	Empty	Noise at $p = 0.13$	All checks fired
3	1,671	19	19–28	60
9	12,561	157	226–318	780
17	41,601	565	823–1,151	4,088
25	87,537	1,229	1,737–2,550	11,492
33	150,369	2,149	3,028–4,560	24,528

For example, the distance-33 patch has 2,113 data qubits. The handoff, including one read per qubit and acceptance of the residual, uses 2,115 cycles. For an empty patch, adding 33 search cycles and one output cycle gives the 2,149 cycles in Table B.5. On the 32 noisy inputs in the table the time after the CNN ranged from 30.28 to 45.60 µs at 100 MHz. On 32 further frames drawn at the same error probability, it ranged from 28.27 to 41.94 µs, and a second simulator reproduced the search time on every residual. The all-fired input took 245.28 µs. A conservative count of all possible row visits and path steps gives an upper bound of 172,301 added cycles for this circuit at distance 33.

The banked CNN and cleanup circuit together met 100 MHz on the VU13P after routing at distance 33. It used 59,383 LUTs, 38,048 flip-flops, 77.5 BRAM36 equivalents and 189 DSPs, with setup and hold margins of +1.858 ns and +0.040 ns on its internal paths. The larger-distance hardware timings apply to HDRG. The distance-3 minimum-weight search would need to evaluate exponentially more masks on a larger patch. Its timing therefore cannot be used as an estimate for distance-33 MWPM. The cleanup circuits and test instructions give the full set of distances and input patterns.

C. Supporting decoding-quality results

The first two experiments, introduced in Section 7.5, test the effect of removing small CNN weights. The rest of the appendix gives the device measurements and qubit layouts behind Chapter 7.

C.1 Pruning with complete decoding

A design that skips zero weights could use fewer multiplications. The first test measured how the failure rate changed when small weights were set to zero. The weights in all three layers were ranked by absolute value and the smallest were set to zero. The biases and the remaining weights were unchanged.

Each pruning level used one floating-point CNN pass followed by MWPM or HDRG. The patches had no internal defects. At $p = 0.13$ there were 32,768 fresh trials at distance 9, 16,384 at distance 17 and 8,192 at distance 33. Figure C.1 compares each hybrid with its global decoder alone, without a CNN. The 0% point therefore measures the effect of adding the full, unpruned network. For example, at distance 33 the MWPM failure rate falls from 5.60% to 4.87%, a change of -0.73 percentage points even though no weights have been removed. Table C.1 gives the failure rates and marks changes from the network with all its weights.

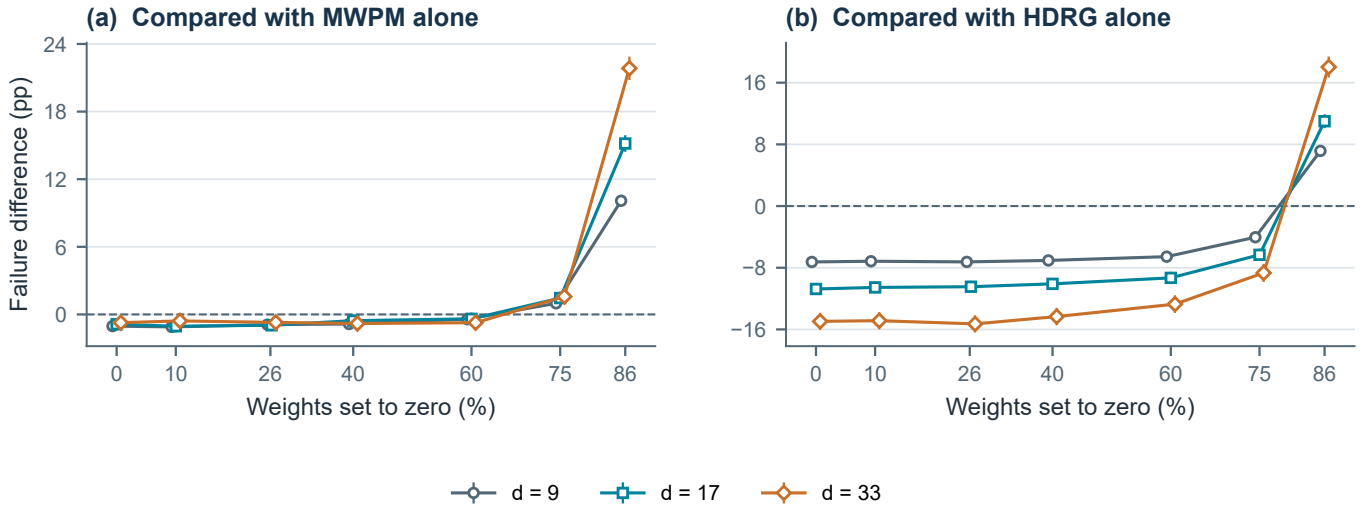


Figure C.1: Magnitude pruning with complete decoding

Change in failure rate from (a) MWPM alone or (b) HDRG alone when one CNN pass is added at $p = 0.13$. The horizontal zero line represents the global decoder without a CNN. At 0% pruning the full network is still present, so its change need not be zero. Values below zero favour the hybrid. Bars show paired 95% confidence intervals. The smallest weights across all three layers were set to zero without retraining. Each of the seven pruning levels uses the same trials within a distance, with 32,768 at distance 9, 16,384 at distance 17 and 8,192 at distance 33. Markers at the same sparsity are offset slightly for readability.

Table C.1: Failure rate (%) of one pruned floating-point CNN pass followed by MWPM (first three columns) or HDRG (last three) at $p = 0.13$ (32,768 at $d = 9$, 16,384 at $d = 17$, 8,192 at $d = 33$), by the fraction of weights set to zero. An asterisk marks a resolved change from the unpruned network (paired 95% interval excluding zero).

Weights set to zero	MWPM $d = 9$	MWPM $d = 17$	MWPM $d = 33$	HDRG $d = 9$	HDRG $d = 17$	HDRG $d = 33$
0%	14.64	10.25	4.87	18.11	15.67	12.16
10%	14.58	10.07	5.03	18.20	15.87	12.24
26%	14.77	10.16	4.90	18.12	15.96	11.83
40%	14.83	10.57	4.80	18.30	16.33*	12.78
60%	15.26*	10.73*	4.87	18.79*	17.11*	14.39*
75%	16.67*	12.57*	7.19*	21.31*	20.10*	18.44*
86%	25.77*	26.29*	27.44*	32.51*	37.43*	45.14*
Cleanup alone	15.68	11.12	5.60	25.36	26.42	27.11

At 26% pruning, the experiment could not distinguish the failure rates from those of the full network. A small difference may still exist. With 60% of the weights removed, adding the CNN to MWPM offered little improvement at distances 9 and 17. At 75%, this combination failed more often than MWPM alone at all three distances. The CNN-plus-HDRG failure rate also increased at high pruning levels. At distance 33, the 86%-pruned hybrid failed on about 45% of trials, compared with 27% for HDRG alone.

The FPGA used the full networks. Its schedule reads every weight, including a zero, so this experiment produced no hardware speedup. A design that skips zeros would also need a way to find the next nonzero weight and its input.

C.2 Local prediction sensitivity with defects

The local test used 200 frames with an error probability of 0.10. The 65×65 grid has the dimensions of the $d = 33$ lattice, but it also contains two 3×3 Z-cut defects. These holes change the paths a logical error can take, so the grid size alone does not give the logical distance. Eight pruning levels were tested on the same frames, with the CNN run once each time without retraining or cleanup.

The first measure counts correction bits that differ from the sampled error. The second compares each prediction with the original network's prediction on that same entry (Figure C.2). At 26.2% pruning, the first count rose slightly, from 26,490 to 26,648. Across 1,690,000 entries, 99.89% of predictions still agreed. At 86.0% pruning, agreement was still 97.23%, yet the difference from the sampled error had more than doubled to 57,323.

Most entries contain no error. Agreement on those entries can conceal differences in the corrections predicted at other sites. Conversely, two different corrections can recover the same logical state. The complete-decoding test therefore checks whether the logical state was recovered, even when individual correction bits differ.

C.3 Final-measurement records from the IBM quantum processor

To estimate how quickly errors accumulated while the qubits were idle (Section 7.3.1), the average frequencies in Figure 7.5 were fitted with $p(t) = \epsilon + A[1 - e^{-(t/T)^{\alpha}}]$. The curve starts at ϵ, rises by A and has a time scale T. The exponent α sets its shape.

For all-one Z memory, the fit with $\alpha = 1$ gave $T = 203 \pm 25$ μs. For X memory it gave 30 ± 4 μs under free evolution and 262 ± 131 μs with the XY4 pulse sequence. The ranges are one standard error of the fit. The longest XY4 delay

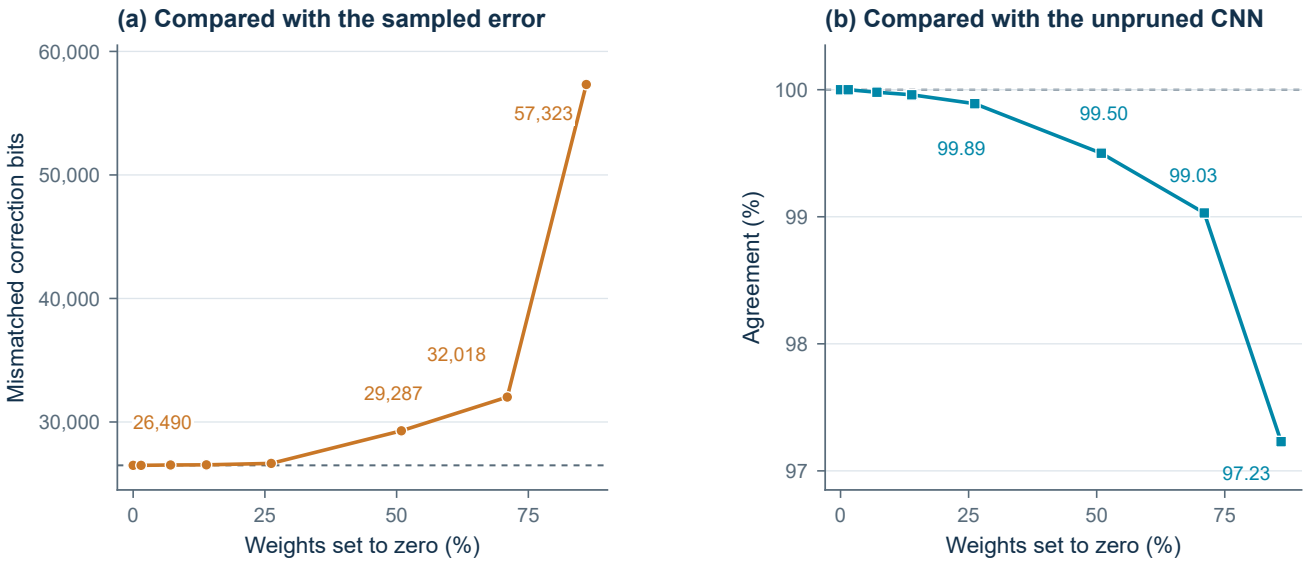


Figure C.2: Effect of pruning on local CNN predictions

One floating-point CNN pass on the same 200 frames at error probability 0.10, using 65×65 grids with a pair of Z-cut defects. Eight thresholds set small weights to zero without retraining. (a) Number of correction bits that differ from the sampled error across both channels. (b) Agreement with the unpruned predictions over 1,690,000 entries, including zero-target positions. The panels measure local prediction changes; complete decoding and hardware speed require separate tests.

measured was 120 µs, so the fitted time scale lies well beyond the measured range. The fit suggests slower error growth, but the fitted time scale remains uncertain. These fits describe the group of measured qubits.

The Bell experiment records bit-like errors, phase-like errors and cases where both occur. If the two errors occurred independently, multiplying their observed frequencies would predict how often both appeared. At zero delay that calculation gives 0.31%, while the measured joint frequency was 1.15%. The ratio was 3.72, with a 95% range of 3.63 to 3.82 when resampling whole shots.

Figure C.3 shows how the relationship changes at later times. At 20 µs the ratio was about 0.91 under free evolution and 1.73 with XY4. One fixed ratio would therefore describe the measured conditions poorly.

C.4 Distance comparison on device measurements

The final-measurement experiments of Section 7.3.1 used patches at distances 5 through 8 (Figure C.4). The standalone integer CNN ran up to eight passes on the first 2,000 shots. A shot counted as a CNN failure if the logical parity was wrong or if any checks remained. MWPM used all 8,192 shots and failed if the logical parity after cleanup was wrong. The figure shows how each method changes with distance. Chapter 7 compares complete decoders using the same failure rule.

A larger patch uses different physical qubits. For all-one Z memory after a 40 µs idle, the observed error frequency per site increased from 13.43% at distance 5 to 16.18% at distance 8. The MWPM failure rate rose from 24.57% to 38.42%. Under its stricter rule, the CNN failure rate rose from 35.70% to 50.25%. Both the patch size and the observed noise changed in this comparison.

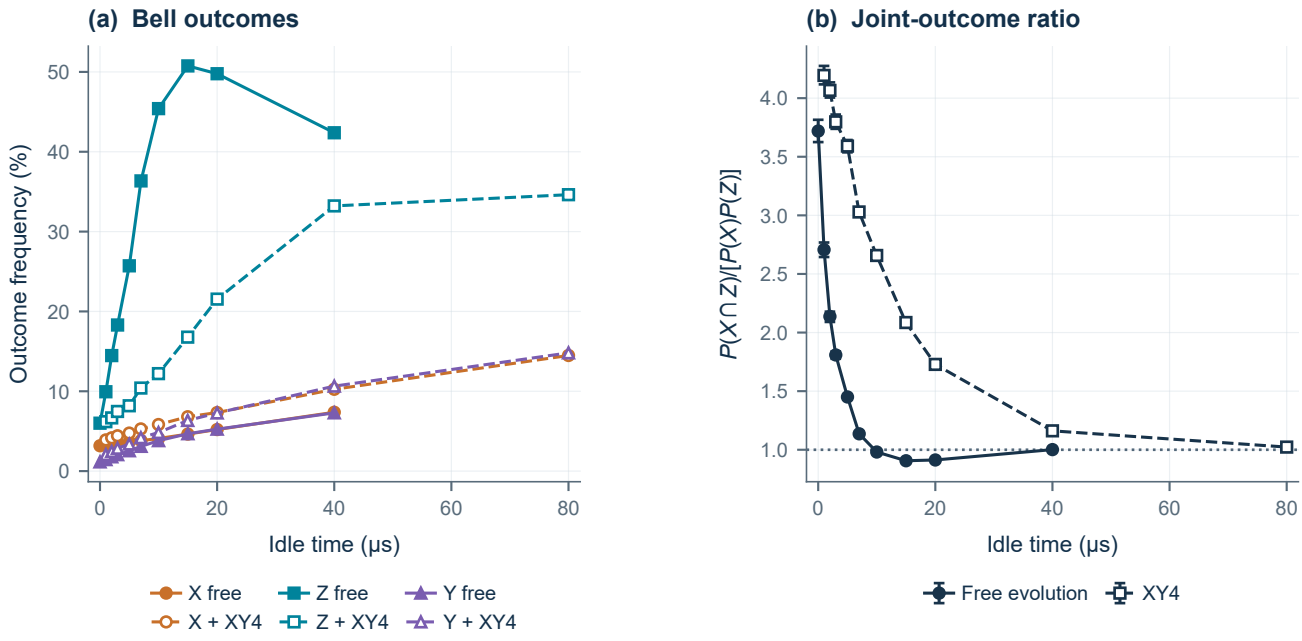


Figure C.3: Bell outcome frequencies and correlations

(a) Frequencies of bit-like only, phase-like only and joint outcomes after Bell-pair preparation and readout, with free evolution (solid) and XY4 (dashed). (b) The joint frequency divided by the product of the overall bit-like and phase-like frequencies; above one, joint errors are more common than independence predicts. The X, Z and Y labels denote bit-like, phase-like and joint outcomes respectively. Error bars are bootstrap 95% ranges over whole shots.

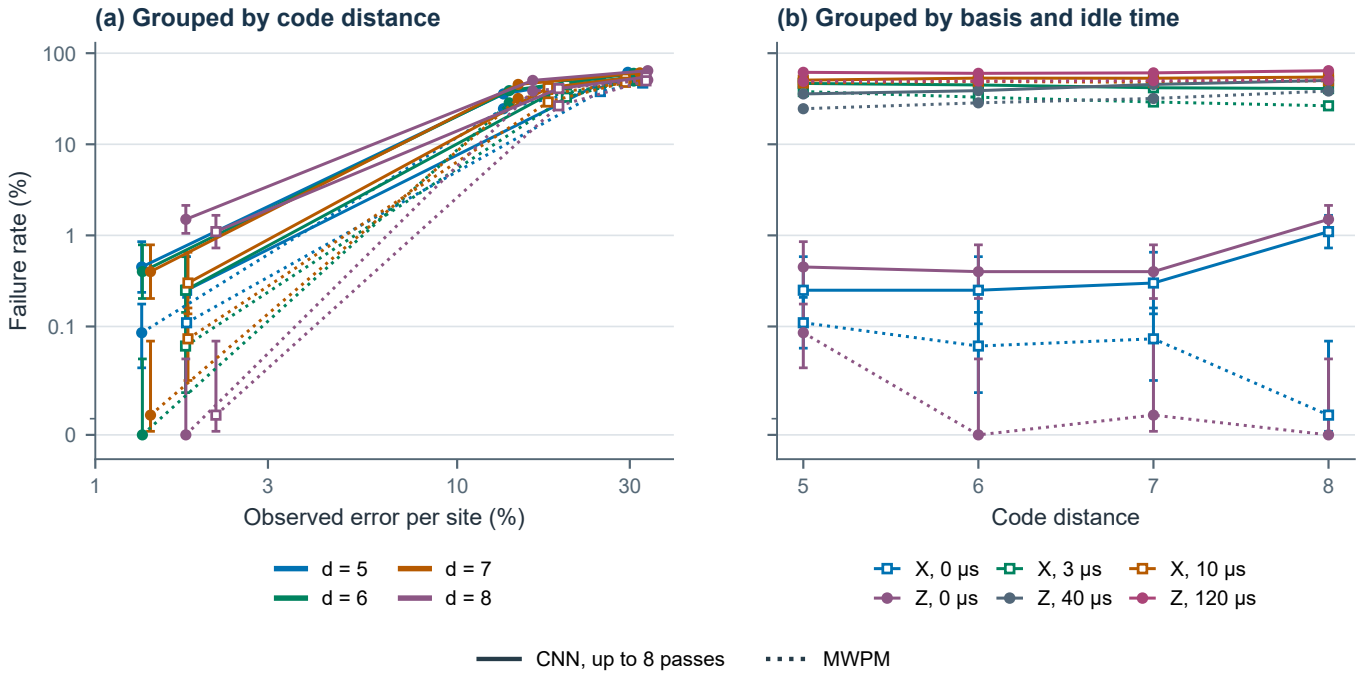


Figure C.4: Decoder failures at code distances 5 to 8

(a) Failure rate against observed error per site, coloured by distance. (b) The same data by distance, grouped by basis and idle time. Squares denote X memory. Z memory (circles) starts all-zero at zero idle and all-one at 40 and 120 µs. The CNN counts wrong parity or uncleared checks as failure (first 2,000 shots, up to eight integer passes). MWPM counts wrong parity (all 8,192 shots). Bars show 95% intervals. The scale is linear below 0.05% and logarithmic above.

At zero idle, MWPM had no observed failures in some sets of 8,192 shots. Even then the 95% upper limit was about 0.047%. Simulation keeps the error model fixed as the patch size changes. On the processor, these numbers describe the particular qubits used.

C.5 The repeated-check records

Figure C.5 gives the four-pass comparison discussed in Section 7.3.2. It plots X and Z memory across one to nine rounds at distance 5.

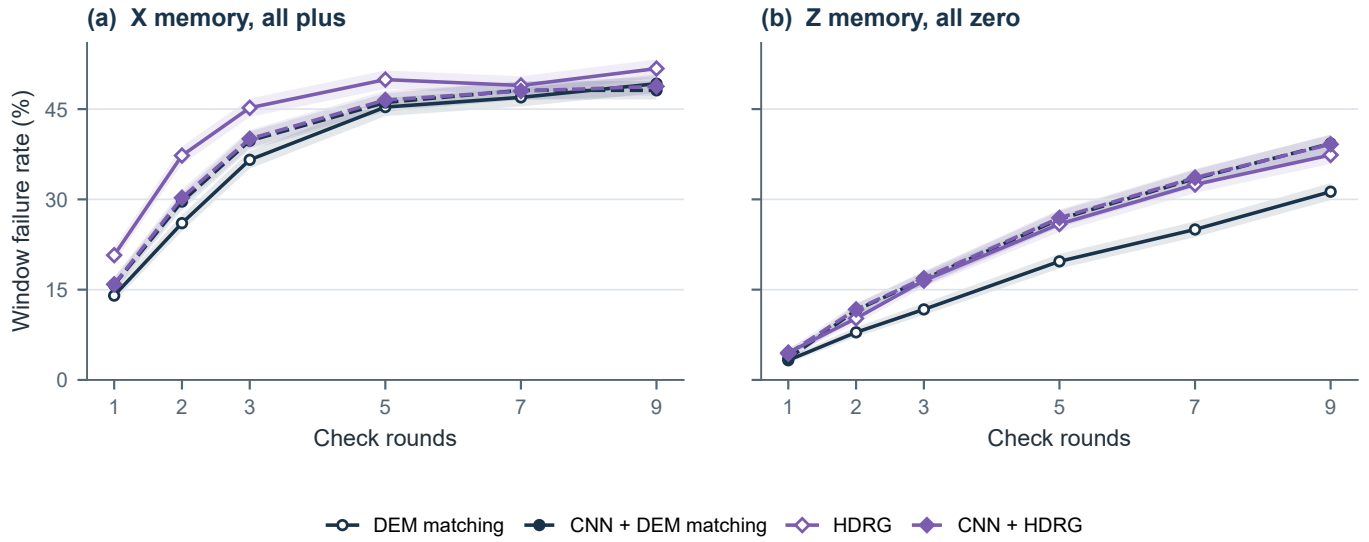


Figure C.5: Decoding repeated-check records from ibm_miami

Distance-5 (a) X-memory and (b) Z-memory experiments with one to nine rounds and 4,096 shots per point. DEM matching and HDRG run alone (solid) and after four floating-point CNN passes (dashed). Shading shows 95% Wilson intervals. All decoding took place offline.

A detector fires when the measurements it compares disagree (Section 2.3). In the middle of these no-reset circuits, it compares readings two rounds apart. One fault can trigger several comparisons, and two changes can cancel in the same comparison. In nine rounds of distance-5 Z memory, about 24% of the available detectors fired (Figure C.6). Across positions in the distance-5 patch, the observed firing rate ranged from about 5.7% to 48.3% of shots (Figure C.7). The average of 24% does not show this variation between positions.

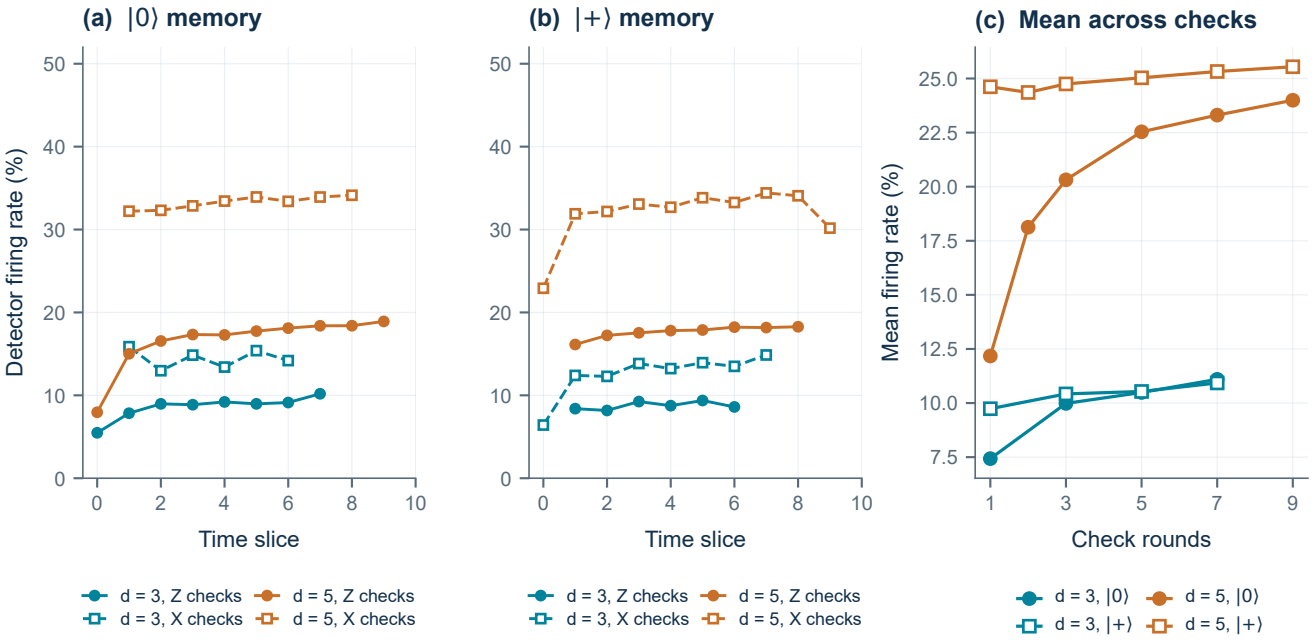


Figure C.6: Detector activity on ibm_miami

Firing rates for X and Z checks in (a) all-zero Z memory, prepared in $|0\rangle$, and (b) plus-state X memory, prepared in $|+\rangle$. Each time slice pools the available checks of that type over 4,096 shots, using the deepest run at each distance. (c) Mean firing rate over all available detectors against the number of rounds. X checks have no known preparation or final-readout reference in Z memory, and Z checks have none in X memory, so those boundary entries are omitted. Missing ancillas and padding are also excluded.

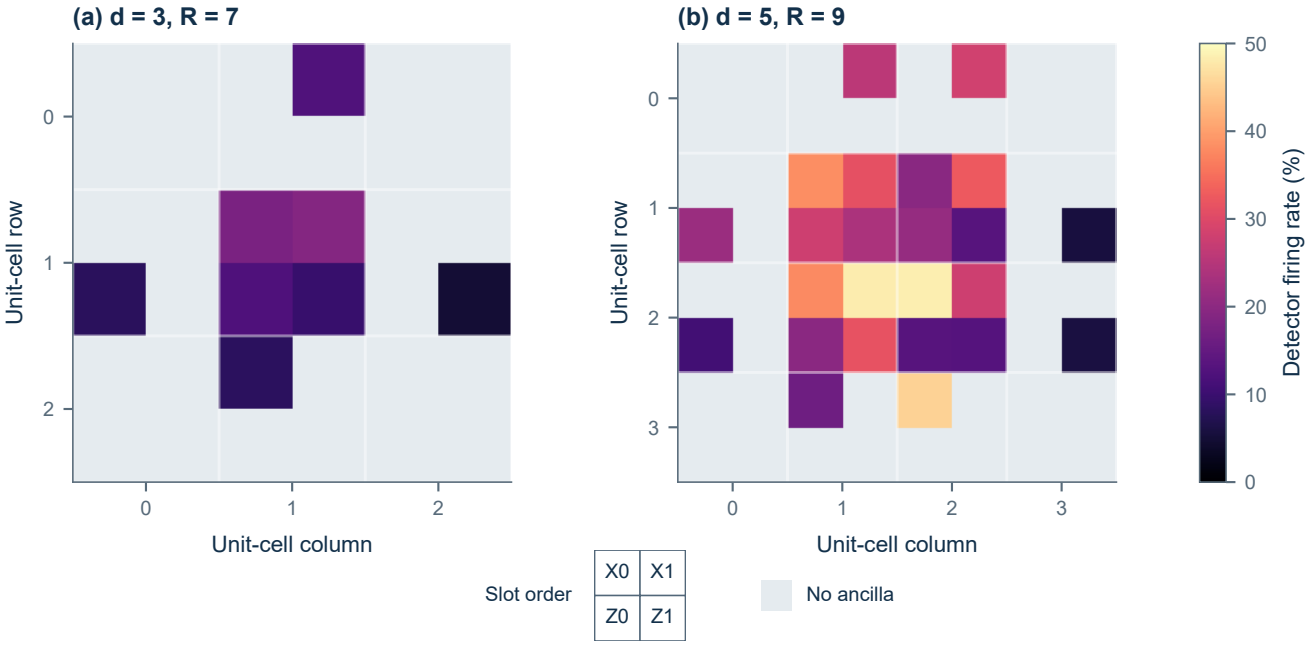


Figure C.7: Detector firing rates across the patch

Detector firing rates in the deepest Z-memory runs, (a) $d = 3, R = 7$ and (b) $d = 5, R = 9$, averaged over all shots and interior time slices. Each unit cell shows four ancilla slots; grey slots have no ancilla. Coordinates are those of the network input, not chip positions.

C.6 Device layouts

At distances 5 through 8 the final-measurement experiments used 41, 61, 85 and 113 data qubits, respectively. The next patch would need 145, more than the 120 qubits in the recorded processor. The Bell experiments paired neighbouring qubits, but the records do not identify which pairs were used.

The repeated-check circuits also used ancillas to measure the checks. Figure C.8 shows their recorded placements. Distance 3 used nine data qubits and eight ancillas. Distance 5 used 25 data qubits and 24 ancillas. The device map identifies which physical qubits can interact using two-qubit gates. The decoder grid arranges the measurements by their positions in the code. The saved placement connects these two descriptions.

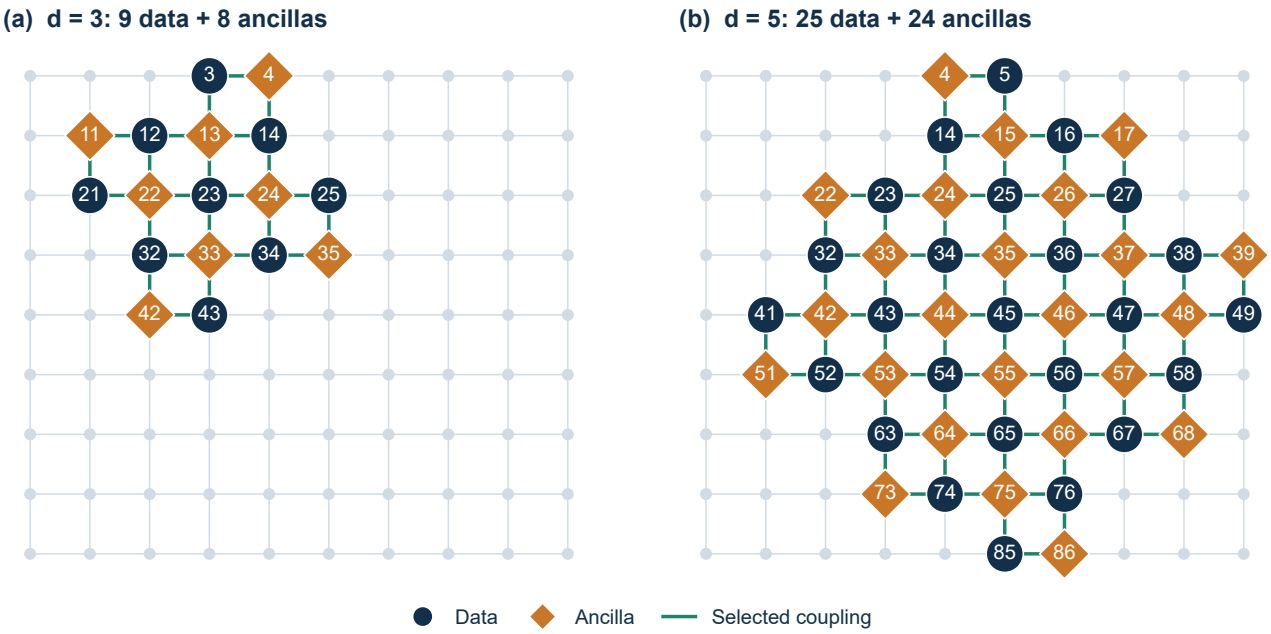


Figure C.8: Recorded qubit placements for repeated checks

The saved circuit placements use (a) nine data qubits and eight ancillas at distance 3 and (b) 25 data qubits and 24 ancillas at distance 5. Navy circles and orange diamonds identify data and ancillas, with physical qubit IDs. Green edges mark selected connections on the recorded chip graph. Connections specify which qubits can interact through two-qubit gates, not the order or timing of those gates.

D. Supplementary validation

Separate checks examined the arithmetic (Section 4.5) and the decoding comparisons (Section 7.1.1). The arithmetic check starts with the weights stored in the FPGA design and verifies the calculated values. The decoding check draws fresh error samples to test whether the earlier comparisons remain consistent.

D.1 Integer and RTL checks

A second NumPy implementation was written for each network. It reads the coefficients in the same packed format as the hardware. For both networks, this independent implementation reproduced the saved output of every layer. This checks that the stored coefficients were read with the intended sign and scale.

The 2D hardware was then run in Icarus Verilog. At each of distances 5, 9, 17 and 33, three frames were sent through in succession, without resetting the core between frames. All 34,104 prediction bits agreed with the integer reference. For the first three distances, the $(11, 4, 6)$ design finished in 4,377, 12,561 and 41,601 cycles. At distance 33, the $(11, 6, 6)$ design took 103,894 cycles and returned 8,450 decisions per frame.

At distance 5 every sum and activation was checked. For the larger patches the final decisions and the cycle counts were checked. The counts agreed with the board results for the same lane settings. The distance-33 simulation used a different setting for assigning arithmetic to DSP blocks, so it provides a check on the schedule rather than a replay of the exact board build.

For the 3D core, the main simulator checked a design that processes 64 hidden output channels together, at distances 5, 9, 17, 25, 33 and 49, and a design with 32 such lanes at distance 49. It ran 42 complete windows. A second simulator checked two more windows at distance 5. Together these tests compared more than 408 million sums and activations.

With 128 output lanes, 36 windows across the same six distances were run in the main simulator, checking more than 247 million values. The second simulator checked two further windows at distance 5. Appendix B.5 gives the hardware costs and timing for the larger designs.

D.2 Fresh-sample decoding check

The 2D one-pass test at $p = 0.13$ from Section 7.1.1 was repeated on fresh samples. The patch layout, error sampling, CNN outputs and matching set-up were all produced independently. This check used the same HDRG implementation. Each decoder saw the same errors within a distance, using new random seeds for this check. There were 65,536 trials at distance 9, 32,768 at distance 17 and 16,384 at distance 33. The integer calculation was also checked against direct convolution on test inputs using the packed coefficients.

Across all 27 comparisons, the largest difference from the earlier failure rate was 2.1 estimated standard errors. The largest improvement from adding the CNN was with HDRG. At distance 33, adding the floating-point CNN reduced failures from 27.26% to 13.13%. For correlated matching, failures rose from 1.43% to 4.38%. MWPM changed much less, from 5.27% to 5.03%.

The paired 95% intervals give the uncertainty in these changes, in percentage points (Section 4.8). For HDRG the change was -14.1 $[-14.9, -13.3]$, and for correlated matching it was $+3.0$ $[+2.6, +3.3]$. For MWPM it was -0.24 $[-0.66, +0.17]$. The MWPM interval includes zero. The fresh sample again showed a benefit for HDRG and more failures for correlated matching. For MWPM, the sample was consistent with a small change in either direction.

Acknowledgements

I acknowledge OpenAI ChatGPT for assistance with debugging and refactoring the Verilog, implementing and testing the FPGA cleanup prototypes, and refining the figures.

I also acknowledge Anthropic Claude for its help while I built Posedge, the FPGA course I used to learn digital design. I later adapted the lessons on timing and memory to lead into the 2D and 3D decoder implementations in this thesis.

This research was supported by the IBM Quantum Hub at the University of Melbourne, which provided access to IBM Quantum processors.