Precision-Pyramid: Towards real-time neural decoding for fault-tolerant quantum computing

Zhenhao Zhong[1,2*] **Ge Yan**[1*†] **Shanchuan Li**[1,3*] **Pengyue Ma**[1] **Yuxuan Du**[1†]

[1]Nanyang Technological University [2]Xi'an Jiaotong University
[3]Tokyo University of Agriculture and Technology

Abstract

Real-time quantum error correction (QEC) is a critical bottleneck for fault-tolerant quantum computing due to strict hardware latency constraints. While neural decoders achieve state-of-the-art logical accuracy, their reliance on high-precision, compute-intensive inference precludes real-time deployment. Conversely, uniformly quantizing these models to extreme low-bit regimes for FPGAs triggers a catastrophic collapse in decoding accuracy. To overcome this precision-accuracy bottleneck, we propose Precision Pyramid (PP), a hardware-algorithm co-designed neural decoder applicable to both surface and BB codes. PP features a monotonically escalating activation precision hierarchy built upon a globally weight-binarized (W1) foundation. By scaling intermediate activations down to INT2 for the massive perception stages and reserving higher precision strictly for the lightweight final logical decision, this hierarchy effectively shifts over 99% of the computational workload to abundant FPGA look-up tables. Comprehensive evaluations on both simulated and hardware-calibrated noise data demonstrate that PP consistently suppresses logical errors below highly optimized classical baselines (e.g., MWPM and Relay-BP) while seamlessly satisfying stringent sub-microsecond latency budgets. Code is available at https://github.com/RockfireTip/Precision-Pyramid.

1 Introduction

Quantum error correction (QEC) is essential for scaling quantum computation beyond noisy physical qubits and ultimately realizing computational advantages [1, 2]. In a QEC code, logical information is encoded nonlocally into many physical qubits, and a classical decoder uses repeated syndrome measurements to infer logical updates or recovery operations without directly measuring the encoded state [3, 4]. Among the many code architectures developed over the past decades, surface codes [5, 6] and quantum low-density parity-check (qLDPC) codes [7, 8, 9] have emerged as two leading candidates for experimental fault tolerance. Surface codes are well matched to two-dimensional local connectivity and have enabled landmark demonstrations of logical error suppression in superconducting processors [10, 11], whereas qLDPC codes, including bivariate bicycle (BB) codes, offer higher encoding rates and are increasingly relevant to hardware platforms with flexible connectivity, such as neutral-atom processors [12, 13]. Recent memory experiments based on these codes have shown that encoded quantum states can be preserved over repeated rounds of syndrome extraction, with logical error rates suppressed as code size increases [14, 15]. Yet decoding in such experiments is largely performed offline, after syndrome data have been collected and with a time budget far beyond that available in an operating quantum processor. This setting is impractical for fault-tolerant (FT) quantum computing, where the decoder must process syndrome measurements as they are

*These authors contributed equally to this work.
†Correspondence to ge.yan@ntu.edu.sg, duyuxuan123@gmail.com.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

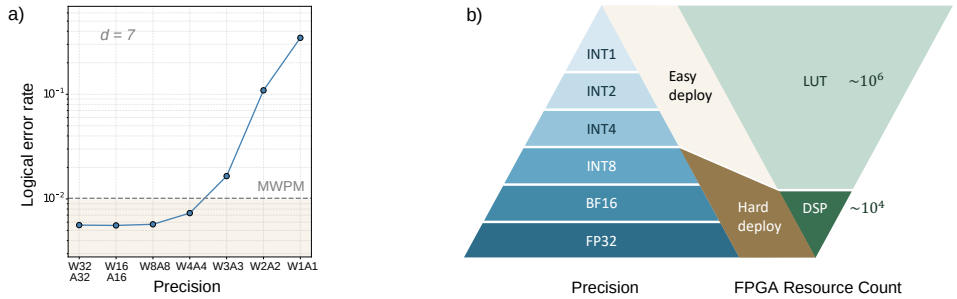


Figure 1: **The intrinsic tension between algorithmic accuracy and hardware constraints in neural decoding. a)** Performance collapse under uniform quantization. Compressing a standard neural decoder uniformly to extreme low-bit regimes ($<$ INT4) for $d = 7$ surface code triggers a catastrophic degradation in the logical error rate. **b)** The conceptual mapping between numerical precision and FPGA resource asymmetry. Low-bit operations can be efficiently implemented using abundant LUTs ($\sim 10^6$) on the FPGA. Conversely, scaling to precisions higher than INT4 necessitates a deployment shift to scarce DSPs (typically $< 10^4$).

produced and determine the corresponding recovery operation within a limited time budget [16]. This necessitates decoding from a purely algorithmic inference problem into an accuracy-latency co-design problem.

To address this accuracy-latency challenge, substantial effort has been devoted to decoder design and deployment. Broadly, existing approaches can be divided into two lines. The first line develops conventional algorithmic decoders, which exploit the structure of specific code families to perform efficient inference. Representative examples include minimum-weight perfect matching (MWPM) [6] for surface codes and belief-propagation-based methods [17] and variants [18, 19] for qLDPC codes. Recent work has further investigated how these decoders can be accelerated on specialized hardware, including field-programmable gate arrays (FPGAs), to better meet real-time constraints [20, 21]. The second line develops neural decoders, which learn decoding rules directly from simulated or experimental syndrome data [22]. When inference latency is not the primary constraint, neural decoders have shown strong decoding performance and, in some settings, can outperform conventional baselines by capturing complex spatiotemporal error correlations [10, 23]. However, real-time neural decoding remains much less explored. A key obstacle is that high-performing neural decoders are often computationally heavy, relying on large parameter counts, high-precision arithmetic, and GPU-based inference. This motivates a decoding-specific view of compression, where numerical precision is co-designed with the QEC task and the target hardware. We therefore ask:

Can a neural decoder achieve real-time FPGA deployment while preserving FT decoding accuracy?

At first glance, the answer may appear straightforward: compress a neural decoder using existing quantization techniques [24, 25] and deploy the resulting low-bit model on an FPGA [26]. However, this naive route breaks down in the regime most relevant to real-time decoding. As shown in Fig. 1a, for surface-code decoding, uniform low-bit quantization causes a severe degradation in decoding performance, falling below the MWPM baseline precisely at the extreme precisions needed for efficient FPGA implementation. These results reflect the precision-accuracy bottleneck, as real-time neural decoding cannot be achieved by naive quantization alone.

To answer this question affirmatively, we propose Precision Pyramid (PP), a real-time neural decoder designed for both surface and BB codes. PP is built around two coupled design principles: *topology-aware syndrome processing and hardware-aware precision allocation*. On the algorithmic side, PP performs syndrome processing on the decoding graph, enabling the same decoding framework to handle both the local two-dimensional geometry of surface codes and the sparse nonlocal connectivity of BB codes. This graph-based processing is combined with causal temporal modeling, allowing PP to respect both the spatial topology of the code and the temporal causality required by real-time decoding. On the hardware side, PP treats numerical precision as an architectural design choice rather than a post-training compression knob. Concretely, it exploits a task-specific asymmetry in QEC decoding. As visualized in Fig. 1b, large-scale spatiotemporal syndrome processing dominates the computational workload and can be executed with ultra-low-bit operations, whereas the final logical decision is more precision-sensitive but accounts for only a small fraction of the computation. PP therefore allocates precision unevenly across the pipeline, from low-bit graph and temporal

perception to higher-precision logical readout. In this way, PP preserves FT decoding accuracy while retaining the parallelism and resource efficiency required for real-time FPGA inference.

We conduct systematic experiments to validate the effectiveness of PP. Our evaluations rely on large-scale Stim simulations alongside the Google Sycamore dataset, which currently stands as the only publicly available error data from physical superconducting hardware. For planar surface codes, we benchmark our decoder up to distance $d = 9$ under two different error models. Encompassing up to 161 physical qubits, this scale inherently covers the frontier of current hardware capabilities and near-term fault-tolerant milestones. Across all configurations, the PP decoder consistently achieves a superior logical error rate, surpassing the classical MWPM baseline. Furthermore, to demonstrate architectural adaptability to emerging platforms such as neutral atoms, we extend our evaluation to the BB code. Here, the PP architecture successfully outperforms the advanced variants of BP. Crucially, these accuracy milestones do not compromise real-time efficiency. Post-route FPGA implementation confirms that the PP decoder executes within the 1 μs latency budget for surface code, guaranteeing continuous, fault-tolerant decoding for fast-cycling quantum processors.

2 Preliminary

This section presents the core concepts of QEC codes and neural decoders, from which we derive the intuition for extreme quantization and FPGA-oriented design. Refer to Appendix A for more details.

QEC Codes. QEC protects quantum information by encoding k logical qubits into n physical qubits, denoted by the parameters $[\![n, k, d]\!]$, where d is the code distance [27]. Among the many QEC codes developed over the past decades, stabilizer codes are particularly prominent because of their well-defined mathematical structure and experimental accessibility. Their central idea is to specify an encoded quantum state through Pauli symmetries. Let $\mathcal{P}_n = \{\pm 1, \pm i\} \times \{I, X, Y, Z\}^{\otimes n}$ denote the n-qubit Pauli group [28]. A stabilizer group $\mathcal{S} \subset \mathcal{P}_n \setminus \{-I\}$ is an Abelian subgroup, and a state $|\psi\rangle$ is stabilized by $\mathcal{S}$ if $S|\psi\rangle = |\psi\rangle$ for all $S \in \mathcal{S}$. An $[\![n, k, d]\!]$ stabilizer code can be specified by $n - k$ independent stabilizer generators, whose simultaneous $+1$ eigenspace defines the code space.

Errors are detected by measuring stabilizers: a physical error that *anticommutes* with a stabilizer flips its measurement outcome, producing a binary **syndrome** $\boldsymbol{s} \in \{0, 1\}^{n-k}$ for one syndrome round. Since different errors can yield the same syndrome, decoding is an inference problem over syndrome histories. QEC code can be represented by a decoding graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes are detectors or stabilizer checks, edges encode syndrome dependencies, and $\mathcal{N}(v)$ denotes the neighborhood used for local aggregation. Surface-code graphs have local two-dimensional lattice connectivity, whereas BB-code graphs are sparse nonlocal Tanner graphs induced by quasi-cyclic parity checks.

Memory-Z experiment. Following standard fault-tolerant conventions [14, 15], we focus on memory-Z experiments, which provide a standard setting for evaluating whether an encoded logical state can be preserved over repeated rounds of noisy syndrome extraction. For an $[\![n, k, d]\!]$ code, the experiment initializes a logical state, performs r rounds of syndrome measurements, typically with $r = d$, and finally checks whether the target logical Z observables have been preserved. The resulting input is a spatiotemporal syndrome history $\mathbf{s} \in \{0, 1\}^{r \times (n-k)}$, and the decoding target is a logical parity vector $\mathbf{y} \in \{0, 1\}^k$. Here, $\mathbf{y}_j = 1$ indicates that the accumulated errors have induced a nontrivial flip of the j-th logical observable. This formulation applies to both surface codes, where typically $k = 1$, and BB codes, where multiple logical parities are decoded jointly.

Neural decoder. Under the memory-Z formulation above, QEC decoding is formulated as predicting a logical parity vector from a syndrome history. Given $\mathbf{s}$, a neural decoder parameterized by $\boldsymbol{\theta}$ outputs $\hat{\mathbf{y}} \in [0, 1]^k$, where $\hat{\mathbf{y}}_j$ estimates the probability that the j-th logical observable has undergone a nontrivial flip. This formulation covers both surface codes and BB codes. Training against the ground-truth vector $\mathbf{y} \in \{0, 1\}^k$ allows the decoder to learn spatiotemporal syndrome patterns that signal logical failures. Different neural architectures and training strategies have been explored [10, 11, 22, 29, 30].

Neural quantization. Neural network quantization mitigates this bottleneck by mapping the continuous learned parameters (weights) and the dynamic intermediate feature maps (activations) into low-bit discrete formats [31]. To explicitly specify these quantization configurations throughout this paper, we adopt the standard 'WxAy' notation, which denotes x-bit precision for weights and y-bit precision for activations. For extreme hardware efficiency, weights can be strictly binarized to a 1-bit domain (e.g., $\{-1, 1\}$) using a simple sign function, anchoring the network in a highly efficient W1

regime. To maintain numerical consistency and stabilize the forward pass, this weight binarization is typically paired with bounded activation functions, such as Hardtanh, which restrict the dynamic range of the intermediate features. While the discrete nature of such extreme quantization natively halts standard backpropagation, this optimization challenge is efficiently bypassed during training by approximating the gradients using the straight-through estimator (STE) [32].

FPGA resources. At the hardware level, FPGAs rely on two primary computational resources: LUTs and DSP slices. LUTs are general-purpose logic elements, which often number in the millions on modern FPGAs, capable of executing bitwise operations and routing within a single clock cycle. In contrast, DSPs are specialized arithmetic units required for complex, high-precision operations such as floating-point or high-bit-width integer multiplications. This kind of asymmetry is reflected in Fig. 1b. Consequently, deploying conventional, full-precision neural networks rapidly exhausts the available DSPs, creating an insurmountable computational bottleneck for sub-microsecond inference. To unlock the full parallel processing potential of FPGAs, aggressive hardware-aware quantization is imperative to transition the computational burden to LUTs.

3 Implementation of Precision Pyramid

This section presents Precision Pyramid (PP), a topology-aware and hardware-aware neural decoder for real-time QEC decoding. We first provide an overview of the learning problem and the architectural decomposition of PP, before detailing its key components in the following subsection.

We formulate neural decoding as the prediction of logical parity from a multi-round syndrome history. Following the notation in Sec. 2, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote the decoding graph associated with a QEC code, where nodes represent detectors or stabilizer checks and edges encode the code-dependent syndrome connectivity. As such, $\mathcal{G}$ specifies which detectors are connected by the code geometry and syndrome-extraction circuit, and therefore provides the structural information used by the PP. Notably, this representation captures the local two-dimensional geometry of surface codes as well as the sparse nonlocal connectivity of BB codes. Given a syndrome history $\mathbf{s} \in \{0,1\}^{r\times|\mathcal{V}|}$ collected over r rounds of syndrome extraction, PP, denoted by $h(\cdot;\mathcal{G},\boldsymbol{\theta})$, predicts a logical parity vector

$$\hat{\mathbf{y}} = h(\mathbf{s};\mathcal{G},\boldsymbol{\theta}) \in [0,1]^k, \tag{1}$$

where $\hat{\mathbf{y}}_j$ estimates the probability that the j-th logical observable has undergone a nontrivial flip.

Given a training set $\mathcal{D} = \{(\mathbf{s}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, where $\mathbf{y}^{(i)} \in \{0,1\}^k$ denotes the ground-truth logical parity label, we optimize the trainable parameters $\boldsymbol{\theta}$ of PP by minimizing the empirical multi-label binary cross-entropy loss. Denote $h_j(\mathbf{s};\mathcal{G},\boldsymbol{\theta})$ as the j-th output probability of PP, we have

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k}\left[\mathbf{y}_j^{(i)}\log h_j(\mathbf{s}^{(i)};\mathcal{G},\boldsymbol{\theta}) + (1-\mathbf{y}_j^{(i)})\log\Big(1-h_j(\mathbf{s}^{(i)};\mathcal{G},\boldsymbol{\theta})\Big)\right]. \tag{2}$$

Architecture overview of PP. To achieve real-time decoding, the goal is not merely to minimize the loss $\mathcal{L}$, but to do so with a decoder that is lightweight enough for FPGA deployment while retaining high accuracy and generalizing across code geometries. This makes the design of h the central challenge: the model must exploit the intrinsic graph structure $\mathcal{G}$ of the underlying code, including the local geometric regularity of surface codes and the sparse nonlocal connectivity of BB codes, while remaining compatible with low-latency hardware execution. To this end, we implement PP as a sequence of topology-, logic-, and hardware-aware modules. As shown in Fig. 2, the forward map is

$$\hat{\mathbf{y}} = h(\mathbf{s};\mathcal{G},\boldsymbol{\theta}) = h_{\mathrm{read}}^{\mathrm{A32}} \circ h_{\mathrm{pool}}^{\mathrm{A4}} \circ h_{\mathrm{perc}}^{\mathrm{A2}}(\mathbf{s};\mathcal{G}), \tag{3}$$

where h_{perc}, h_{pool}, and h_{read} denote the perception, pooling, and readout modules, respectively, and the superscripts, i.e., A2, A4, and A32, indicate the activation precision used in each stage. All trainable weights in $\boldsymbol{\theta}$ are binarized throughout the network.

Owing to this $\mathrm{A2}\to\mathrm{A4}\to\mathrm{A32}$ precision hierarchy, PP assigns precision according to each stage's computational cost and decision sensitivity. Most syndrome processing is performed in an ultra-low-bit regime, while higher precision is used only near the final logical readout. This non-uniform allocation avoids the accuracy collapse caused by uniform extreme quantization and aligns with the FPGA resource asymmetry illustrated in Fig. 1: low-bit operations can be massively parallelized using abundant LUTs, whereas scarce arithmetic units are reserved for the compact readout. As a result, PP

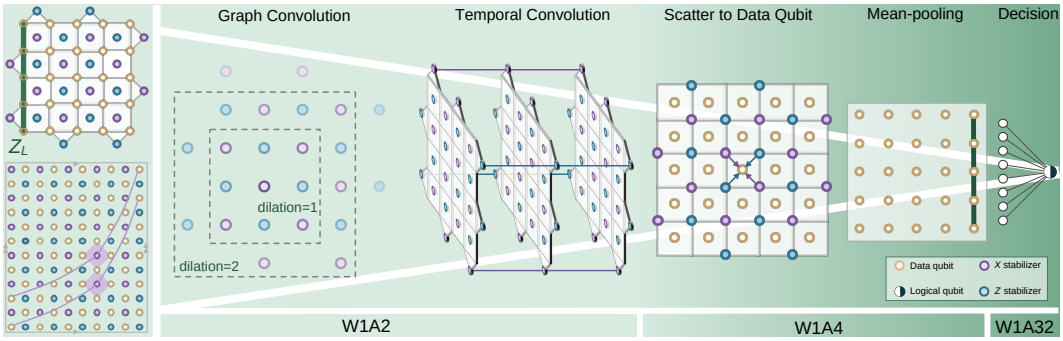


Figure 2: **The** PP **decoding pipeline. (Left)** Schematic representations of the underlying topological graphs, illustrating native compatibility with both planar surface codes and the flattened periodic structure of non-local BB codes. **(Right)** Forward inference trajectory. The architecture globally enforces 100% binarized weights (W1). First, the spatiotemporal perception stage extracts features via spatial graph convolutions (dilated for surface codes) and causal temporal convolutions utilizing 2-bit activations. Subsequently, these latent features are scattered back to the data qubits and aggregated via an operator-aware mean pooling over the canonical support of the target observable (e.g., Z_L for memory-Z), transitioning the activation precision to 4-bit. Finally, a full-precision (32-bit) global decision layer infers the logical parity. This progressively escalating activation hierarchy across the three stages is formally denoted as PP (A2-A4-A32).

is capable of achieving FT decoding accuracy while meeting the parallelism and resource-efficiency requirements of real-time FPGA inference.

We next describe the construction and training of the three modules in Eq. (3): the topology-aware graph-temporal perception module $h^{\mathrm{A2}}_{\mathrm{perc}}$, the logic-aware pooling module $h^{\mathrm{A4}}_{\mathrm{pool}}$, and the high-precision logical readout module $h^{\mathrm{A32}}_{\mathrm{read}}$. The focus is on how these modules exploit the code graph $\mathcal{G}$ while allocating numerical precision according to each stage's computational and statistical role. Additional implementation details are deferred to Appendix C.

3.1 Implementations of key modules and the optimization of PP

Construction of low-bit graph-temporal perception $h^{\mathrm{A2}}_{\mathrm{perc}}$. The role of this module is to transform the syndrome stream $\mathbf{s}$ into latent detector features while respecting both the *spatial topology* of the code and the *temporal causality* required by real-time decoding. Specifically, for each detector node $v \in \mathcal{V}$ and syndrome round t, we first construct an initial embedding $\mathbf{h}^{(0)}_{v,t}$ from the local syndrome pattern around v. Since the neighborhood structure is specified by the decoding graph $\mathcal{G}$, the same rule applies to both the local lattice of surface codes and the sparse nonlocal graph of BB codes.

Starting from the initial detector embeddings, $h^{\mathrm{A2}}_{\mathrm{perc}}$ builds higher-level detector representations by alternating graph-based spatial aggregation and causal temporal aggregation at the stage level, with a $\mathsf{temporal}-\mathsf{spatial}-\mathsf{temporal}$ arrangement. Its output is the collection of final-layer detector features

$$h^{\mathrm{A2}}_{\mathrm{perc}}(\mathbf{s};\mathcal{G}) = \{\mathbf{h}^{(L)}_{v,t} : v \in \mathcal{V},\ t = 1,\dots,r\}, \tag{4}$$

where $\mathbf{h}^{(\ell)}_{v,t}$ denotes the output feature of the ℓ-th graph-temporal layer at detector v and round t. Because $h^{\mathrm{A2}}_{\mathrm{perc}}$ processes the full syndrome history and dominates the computational workload, all activations are kept in the ultra-low-bit A2 regime.

The construction of $\mathbf{h}^{(\ell)}_{v,t}$ contains two steps. First, a spatial step propagates detector features along the decoding graph $\mathcal{G}$, thereby injecting a topology-aware inductive bias. For surface codes, $\mathcal{N}(v)$ corresponds to local neighbors on a two-dimensional lattice; for BB codes, it follows the sparse nonlocal Tanner graph. More specifically, for each detector node $v \in \mathcal{G}$, the model aggregates features from its graph neighborhood $\mathcal{N}(v)$ and applies a learnable transformation with parameters $W^{(\ell)}_{\mathrm{s}}$, followed by a nonlinear activation and 2-bit activation quantization Q_{A2}. This gives the intermediate feature $\mathbf{z}^{(\ell)}_{v,t} = Q_{\mathrm{A2}}[\sigma(\sum_{u\in\mathcal{N}(v)} W^{(\ell)}_{\mathrm{s}}\mathbf{h}^{(\ell-1)}_{u,t})]$, where $W^{(\ell)}_{\mathrm{s}}$ is included in $\boldsymbol{\theta}$.

Second, a causal temporal step aggregates the spatially updated features across syndrome rounds, thereby enforcing the temporal structure required by real-time decoding. That is, for each detector node v and round t, the model combines features only from the current and previous $K-1$ rounds,

followed by a nonlinear activation and 2-bit activation quantization Q_{A2}. This gives the output feature $\mathbf{h}_{v,t}^{(\ell)} = Q_{\mathrm{A2}}[\sigma(\sum_{\tau=0}^{K-1} W_{\mathrm{t},\tau}^{(\ell)} \mathbf{z}_{v,t-\tau}^{(\ell)})]$, where K is the temporal kernel size and $W_{\mathrm{t},\tau}^{(\ell)}$ denotes the trainable temporal transformation at delay τ, with all such parameters collected in $\boldsymbol{\theta}$. Terms with $t-\tau < 1$ are masked or padded. The one-sided summation over $\tau \geq 0$ ensures that the representation at round t never uses future syndrome measurements.

Construction of logic-aware pooling $h_{\mathrm{pool}}^{\mathrm{A4}}$. The perception module $h_{\mathrm{perc}}^{\mathrm{A2}}$ produces detector-level features, whereas the decoding target in Eq. (1) is a logical parity vector. The role of $h_{\mathrm{pool}}^{\mathrm{A4}}$ is therefore to convert local detector information into compact logical representations before the final readout.

To this end, PP first scatters detector features to data qubits using a separate check-to-data incidence relation defined by the parity-check matrices H_X and H_Z. Let $\mathcal{Q}$ denote the set of data qubits and let $\mathcal{N}(q)$ denote the detectors adjacent to data qubit q. For each $q \in \mathcal{Q}$, we aggregate nearby detector features as

$$\mathbf{r}_q = Q_{\mathrm{A4}}\Big[\sigma\big(\sum_{v\in\mathcal{N}(q)} W_{\mathrm{scat}}\, \mathbf{h}_v^{(L)}\big)\Big], \tag{5}$$

where W_{scat} is a trainable linear transformation and Q_{A4} denotes 4-bit activation quantization. We assign A4 precision as it lies closer to the logical decision than the perception module, yet still performs structured aggregation over many features. In this way, $\mathbf{r}_q$ is a data-qubit representation that summarizes nearby detection events and their temporal context. Next, for each logical observable j, PP pools the data-qubit features associated with that observable. Let $\mathcal{Q}_j \subseteq \mathcal{Q}$ denote this data-qubit set. We define $\mathbf{r}_j = \sum_{q\in\mathcal{Q}_j} \mathbf{r}_q/|\mathcal{Q}_j|$.

This logic-aware pooling avoids feeding all detector features into a large global classifier. Instead, it summarizes the evidence relevant to each logical parity into a fixed-dimensional representation $\mathbf{r}_j$, reducing the cost of the final decision while preserving information needed to identify logical flips.

Construction of high-precision logical readout $h_{\mathrm{read}}^{\mathrm{A32}}$. Given the logical representations $\{\mathbf{r}_j\}_{j=1}^{k}$, where k is the number of logical qubits, the readout module $h_{\mathrm{read}}^{\mathrm{A32}}$ predicts the logical-flip probability for each logical observable:

$$\hat{\mathbf{y}}_j = \sigma_{\mathrm{out}}\left(W_{\mathrm{read}}\mathbf{r}_j + b_{\mathrm{read}}\right), \qquad j = 1, \ldots, k. \tag{6}$$

Here, W_{read} and b_{read} are trainable readout parameters, and σ_{out} is the full-precision sigmoid function. Collecting these entries gives the predicted logical parity vector $\hat{\mathbf{y}} \in [0,1]^k$ used in Eq. (2).

Training with fake-end supervision and quantization curriculum. Optimizing PP involves two distinct difficulties. The first difficulty comes from the weak supervision provided by the final logical label $\mathbf{y}$. In a memory experiment, $\mathbf{y}$ only records whether the full r-round syndrome history produces a logical flip; it does not reveal which intermediate rounds are informative or how partial syndrome histories should be interpreted. Since different error histories can share the same final parity, and multiple errors may cancel at the logical level, this supervision is delayed and ambiguous. The second difficulty comes from quantization. With binarized weights and low-bit activations, direct training from random initialization is unstable due to the gradient mismatch between discrete forward computation and continuous backpropagation.

PP addresses the first difficulty using *fake-end supervision*. For an intermediate round $t < r$, fake-end supervision truncates the syndrome history at t, assumes a zero-error future, and queries a classical decoder to produce an intermediate logical-parity target. This construction provides auxiliary teacher signals for partial syndrome histories without requiring intermediate measurements of data qubits. During training, the weight of the fake-end loss is annealed, so that early optimization benefits from teacher-guided supervision, whereas later optimization focuses on the true labels in Eq. (2).

PP addresses the second difficulty using a *progressive quantization curriculum*. The model is first trained in a higher-precision configuration and is then gradually moved toward the target $\mathrm{A2}\to\mathrm{A4}\to\mathrm{A32}$ hierarchy, with each lower-precision model initialized from the converged parameters of the previous stage. Gradients through the quantizers are estimated using the straight-through estimator. Together, fake-end supervision and progressive quantization provide a stable optimization path for training the final FPGA-oriented decoder.

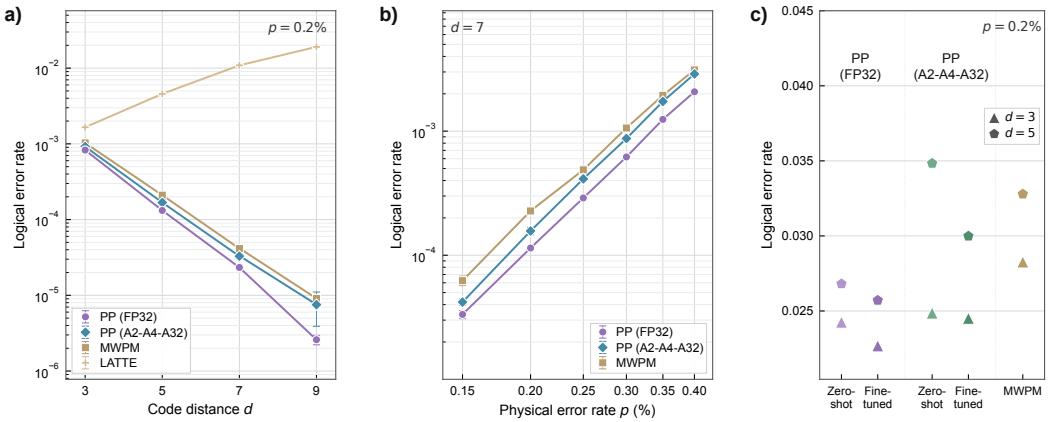


Figure 3: **Decoding performance of the PP architecture on rotated surface code. a)** Logical error rate scaling over code distance d under uniform depolarizing noise ($p = 0.002$). Elevating the activation precision to 2-bit (W1A2) allows PP to successfully surpass the MWPM baseline, rescuing the performance drop of the purely binarized W1A1 model. **b)** Zero-shot evaluation under the asymmetric SI1000 noise model ($d = 7$). PP maintains a consistent advantage over MWPM across varying physical error rates. **c)** Validation on the Google Sycamore hardware dataset ($d = 3, 5$). PP effectively outperforms traditional graph-based decoders against real-world unmodeled noise, with or without empirical fine-tuning.

4 Experiment results

In this section, we systematically evaluate the proposed PP decoder on the rotated surface code with distances $d \in \{3, 5, 7, 9\}$ and BB code [[72,12,6]]. Following conventions [33], we fix the syndrome measurement rounds as $r = d$ and focus on the memory-Z experiment. All the simulation data are generated with Stim [34] under different noise models (as illustrated in Sec. A.2). All experiments were conducted on a server equipped with 8 NVIDIA A100 GPUs, each with 40 GB memory, and an AMD EPYC 7763 CPU. The large-scale Stim simulations were primarily performed on CPU workers, while neural decoder training and evaluation were accelerated on the A100 GPUs.

4.1 Results on surface codes

Experimental setup. We utilize two distinct simulated detector error models (DEMs), i.e., the uniform circuit-level depolarizing noise and the SI1000 model, alongside one hardware-calibrated DEM dedicated to evaluating the Google Sycamore dataset. The PP models are trained at physical error rate $p = 0.005$ for the uniform DEM and $p = 0.004$ for the SI1000 DEM, and then further zero-shot inferred on the reported physical error rates. This is due to the fact that altering p merely changes the appearance probabilities of various syndrome patterns, as long as p does not exceed the transition point where further increasing p would lead to complete chaos. Finally, to ensure a fair and standardized comparison across different code distances where the total number of syndrome extraction rounds varies, we adopt the logical error rate per round as our primary evaluation metric [10, 11]. The omitted details are provided in Appendix D.

Baseline decoders. The conventional baseline is MWPM, the gold standard decoder for 2D surface codes. In our experiments, we utilize PyMatching V2 [35], a state-of-the-art and highly optimized C++ implementation of MWPM. For the FPGA-based neural decoder, we introduce a recent paper LATTE [36], which implements an INT8 neural decoder that fits within the FPGAs. Refer to Appendix D.3 for more details about the implementation of these baselines.

Simulation results. The decoding performance on planar surface codes is systematically evaluated across two distinct noise environments. Fig. 3a illustrates the logical error rate (ϵ_L) as a function of the code distance $d \in \{3, 5, 7, 9\}$ under uniform circuit-level depolarizing noise at a fixed physical error rate of $p = 0.002$. To rigorously validate our precision design, we evaluate both the unquantized theoretical upper bound (FP32, utilizing W32A32) and our highly compressed architecture, PP (A2-A4-A32) (strictly 1-bit weights with progressive activations). As expected, the FP32 model establishes the algorithmic ceiling, significantly outperforming MWPM due to its unconstrained representational capacity. Crucially, PP (A2-A4-A32) largely preserves this advantage under binarized weights, achieving lower logical error rates than MWPM at all evaluated code distances. Ablation studies demonstrating that further precision reduction in any stage of the PP architecture degrades accuracy

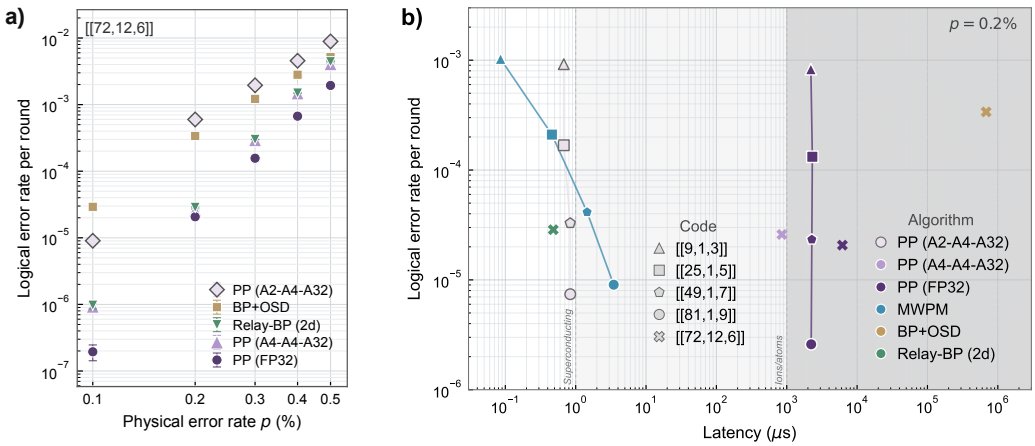


Figure 4: **Comprehensive evaluation of decoding accuracy and hardware execution latency. a)** Decoding performance on the $[\![72, 12, 6]\!]$ BB code. The y-axis reports the normalized LER *per round per logical qubit* to account for the multi-qubit encoding ($k = 12$). While the baseline PP (A2-A4-A32) falls short of BP+OSD due to non-local connectivity demands, moderately relaxing the initial perception precision to 4-bit allows the PP (A4-A4-A32) model to successfully surpass the state-of-the-art Relay-BP baseline. **b)** Latency-accuracy trade-offs under uniform circuit-level noise ($p = 0.2\%$). For planar surface codes, PP (A2-A4-A32) strictly satisfies the sub-microsecond latency budgets of superconducting hardware. For BB codes, PP (A4-A4-A32) inherently incurs higher execution latency but remains comfortably bounded within the operational thresholds of targeted ion- and atom-based platforms.

below this MWPM baseline are detailed in Appendix E.1. In stark contrast, the prior baseline neural decoder, LATTE, exhibits an anomalous upward trajectory, fundamentally failing to suppress errors as the code distance scales.

To further investigate the decoder's robustness under realistic hardware constraints, Fig. 3b evaluates the models under the asymmetric, superconducting-inspired SI1000 noise model at a fixed distance $d = 7$. The results exhibit a consistent hierarchical performance identical to the uniform noise setting. The strictly weight-binarized PP (A2-A4-A32) demonstrates strong generalization, maintaining its decisive advantage over MWPM across the entire evaluated spectrum. This implies that the structural design of the PP is not overfitted to specific symmetric noise or training conditions, but intrinsically capable of resolving complex, hardware-native physical errors at ultra-low hardware cost.

Real hardware data results. To rigorously validate our findings on physical quantum hardware, we evaluate the PP decoder using the publicly available dataset from the Google Sycamore processor. This dataset comprises measurement outcomes from multi-round memory-Z experiments scaling up to 49 physical qubits ($d = 3$ and $d = 5$), containing 200,000 and 50,000 experimental shots, respectively. Crucially, the repository provides a hardware-calibrated DEM fitted directly to the experimental observables. We utilize this DEM to generate Stim-based training datasets. Consequently, we report the performance of the PP decoder under two settings: zero-shot inference (trained exclusively on the DEM-simulated data) and after fine-tuning on the empirical experimental shots. The baseline LERs are directly sourced from the original paper [10].

The experimental results, plotted in Fig. 3c, demonstrate the superior decoding capability of the PP architecture, effectively bridging the gap between idealized simulations and practical hardware deployments. Notably, the PP decoder consistently outperforms the standard MWPM algorithm across both code distances, even in the zero-shot setting without any exposure to the actual experimental data. Likely constrained by the limited volume of the empirical dataset, the subsequent fine-tuning procedure yields only marginal improvements. Most importantly, the performance gap between the PP decoder and its classical counterparts is considerably more pronounced on the real-world Sycamore data than in the preceding pure simulation environments. This compelling outcome indicates that, despite operating at extremely low activation precisions with strictly binarized weights, our neural architecture successfully captures complex, unmodeled physical correlations among qubits, thereby delivering significantly enhanced robustness against real hardware noise compared to conventional graph-based decoders.

4.2 Results on BB codes

Experimental setup. To demonstrate that the proposed PP architecture is not restricted to 2D planar lattices, we extend our evaluation to the BB code, specifically the $[\![72, 12, 6]\!]$ configuration still with $r = d = 6$. Unlike surface codes, BB codes feature a quasi-cyclic torus topology characterized by non-local stabilizer connections and a substantially higher encoding rate ($k = 12$). This evaluation is conducted exclusively under the circuit-level uniform depolarizing noise model. We omit the SI1000 model here, as it is heavily specialized for superconducting architectures; conversely, the non-local connectivity of BB codes makes them inherently better suited for emerging platforms like trapped ions or neutral atoms, for which a standardized, hardware-calibrated DEM is currently unavailable. To ensure a fair comparison across codes with varying logical capacities, we adopt the normalized logical error rate per round and per logical qubit (see Appendix D.1 for details).

Baseline decoders. The complex topology of BB codes renders traditional graph-based matchers like MWPM entirely inapplicable. Belief propagation (BP) serves as the foundational classical decoder for such qLDPC codes. To ensure the most rigorous comparison, we benchmark against two advanced BP variants: BP+OSD and RelayBP [37]. Ordered statistics decoding (OSD) is a sophisticated post-processing technique utilizing Gaussian elimination on the most reliable bits; while it consistently yields superior accuracy, it incurs a severe computational overhead that significantly increases decoding latency. Alternatively, Relay-BP employs a two-stage decoding pipeline where a first-stage relay algorithm dynamically triggers multiple parallel successor BPs. Its ultimate accuracy and decoding latency are heavily dependent on the number of deployed successor BPs and the degree of hardware parallelization. For a fair comparison, both baseline approaches are capped at a standard maximum of $2d$ BP iterations, with comprehensive hyperparameter settings detailed in Appendix D.3.

Simulation results. The decoding performance on the torus topology is systematically evaluated in Fig. 4a, revealing a clear accuracy hierarchy among the evaluated methods. We initially observe that the baseline PP (A2-A4-A32) configuration, which proved optimal for planar surface codes, falls short of the classical BP+OSD baseline. This performance gap indicates that the intricate, non-local connectivity of BB codes inherently demands a higher representational capacity at the initial perception stage. Meanwhile, the advanced Relay-BP algorithm establishes a highly aggressive classical accuracy bound, noticeably outperforming both BP+OSD and our 2-bit perception model. To eclipse this state-of-the-art classical solver, we moderately relax the neural decoder's precision constraints by upgrading the initial spatiotemporal perception stage to 4-bit activations. The resulting PP (A4-A4-A32) configuration successfully suppresses logical errors below Relay-BP, marching toward the absolute algorithmic ceiling established by the unquantized full-precision model (FP32). Although Relay-BP is a highly optimized and strong accuracy baseline, PP remains comparable to and slightly better than it after quantization despite using strictly binary weights and 4-bit activations in the initial spatiotemporal perception stage; while Relay-BP has lower reported latency, the implemented PP decoder still remains within the millisecond-scale QEC-cycle budget relevant to BB-code platforms. This result demonstrates that the main benefit of PP is preserving competitive BB-code accuracy under extreme hardware-oriented quantization on the relevant QEC platforms. Crucially, this targeted precision relaxation represents a highly practical hardware trade-off: 4-bit activations remain natively implementable via LUTs, ensuring that over 99% of the computational workload continues to rely on highly efficient LUT operations rather than expensive DSPs.

4.3 Decoding latency

To rigorously assess real-world hardware efficiency, the execution latency of the highly quantized PP architectures is obtained from post-route implementation in Vivado 2025.2, targeting an AMD Versal Premium VP1902 FPGA under a 400 MHz clock constraint (see Appendix E.6). For the classical baselines, the decoding time of Relay-BP is projected based on the per-iteration FPGA runtime reported in the original literature [19]. Conversely, the unquantized FP32 neural models and all other conventional graph-based decoders are profiled using standard CPU and GPU execution environments.

The global latency-accuracy trade-offs, illustrated in Fig. 4b, demonstrate a definitive architectural advantage. For planar surface codes, PP (A2-A4-A32) comfortably suppresses logical errors below the MWPM baseline while strictly adhering to the stringent sub-microsecond decoding budgets mandated by superconducting devices. For the complex BB codes, PP (A2-A4-A32) achieves microsecond-level execution, maintaining high accuracy. While upgrading to the PP (A4-A4-A32) configuration incurs a slight latency overhead to successfully eclipse Relay-BP's accuracy,

its execution time remains well within the forgiving coherence limits of trapped ion and neutral atom platforms. Consequently, these findings decisively answer the central question posed in our introduction: neural decoders can indeed survive extreme quantization, delivering state-of-the-art logical accuracy while seamlessly satisfying diverse, platform-specific hardware latency constraints.

5 Conclusion

In conclusion, this paper demonstrates that quantum neural decoders are intrinsically resilient to extreme weight quantization, successfully operating under a strictly binarized (100% W1) regime. Building upon this, we proposed the PP architecture. Our evaluations prove that deploying a progressively escalating activation hierarchy—rather than a homogeneous precision configuration—maximizes computational efficiency while effectively capturing complex spatiotemporal physical correlations. Ultimately, our findings confirm that the proposed PP decoder succeeds in surpassing conventional graph-based decoders in accuracy while strictly complying with the stringent decoding latency limitations of modern quantum hardware. By harmonizing these dual objectives, the PP architecture provides a practical, hardware-efficient blueprint for scaling fault-tolerant quantum computing.

6 Limitations

One limitation of our current study lies in the hardware evaluation methodology. While we conduct rigorous HLS and Vivado analyses to systematically validate the sub-microsecond execution of our architecture, the system has not yet been deployed and tested on physical FPGA hardware.

Another limitation concerns scalability beyond the evaluated regime. Although the heterogeneous quantization scheme allows the $d = 9$ surface-code decoder to fit on the VP1902 FPGA, its high LUT utilization leaves limited headroom for larger code distances. Scaling to larger surface-code distances or more complex code families will require more resource-efficient neural decoders or alternative hardware strategies.

Acknowledgment

This study was supported by the Agency for Science, Technology and Research (A*STAR) under Grant No. H25-MRO3488.

References

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

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

[3] Peter W Shor. Fault-tolerant quantum computation. In *Proceedings of 37th conference on foundations of computer science*, pages 56–65. IEEE, 1996.

[4] John Preskill. Reliable quantum computers. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):385–410, 1998.

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

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

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

[8] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005):778–782, 2024.

[9] Pavel Panteleev and Gleb Kalachev. Asymptotically good quantum and locally testable classical ldpc codes. In *Proceedings of the 54th annual ACM SIGACT symposium on theory of computing*, pages 375–388, 2022.

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

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

[12] Dolev Bluvstein, Harry Levine, Giulia Semeghini, Tout T Wang, Sepehr Ebadi, Marcin Kalinowski, Alexander Keesling, Nishad Maskara, Hannes Pichler, Markus Greiner, et al. A quantum processor based on coherent transport of entangled atom arrays. *Nature*, 604(7906):451–456, 2022.

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

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

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

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

[17] David Poulin and Yeojin Chung. On the iterative decoding of sparse quantum codes. *arXiv preprint arXiv:0801.1241*, 2008.

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

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

[20] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 65–74, 2017.

[21] Ben Barber, Kenton M Barnes, Tomasz Bialas, Okan Buğdaycı, Earl T Campbell, Neil I Gillespie, Kauser Johar, Ram Rajan, Adam W Richardson, Luka Skoric, et al. A real-time, scalable, fast and resource-efficient decoder for a quantum computer. *Nature Electronics*, 8(1):84–91, 2025.

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

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

[24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[25] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International conference on machine learning*, pages 7197–7206. PMLR, 2020.

[26] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pages 291–326. Chapman and Hall/CRC, 2022.

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

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

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

[30] Boris M Varbanov, Marc Serra-Peralta, David Byfield, and Barbara M Terhal. Neural network decoder for near-term surface-code experiments. *Physical Review Research*, 7(1):013029, 2025.

[31] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[32] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[33] Andi Gu, J Ataides, Mikhail D Lukin, and Susanne F Yelin. Scalable neural decoders for practical fault-tolerant quantum computation. *arXiv preprint arXiv:2604.08358*, 2026.

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

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

[36] Kai Zhang, Jubo Xu, Fang Zhang, Linghang Kong, Zhengfeng Ji, and Jianxin Chen. Latte: A decoding architecture for quantum computing with temporal and spatial scalability. *arXiv preprint arXiv:2509.03954*, 2025.

[37] Tristan Müller and contributors. Relay-BP: A rust implementation with python bindings for qldpc decoding. `https://github.com/trmue/relay`, 2025. Accessed: 2026-05-07.

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

[39] Hector Bombin and Miguel Angel Martin-Delgado. Topological quantum distillation. *Physical review letters*, 97(18):180501, 2006.

[40] Antonio deMarti iOlius, Patricio Fuentes, Román Orús, Pedro M Crespo, and Josu Etxezarreta Martinez. Decoding algorithms for surface codes. *Quantum*, 8:1498, 2024.

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

[42] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

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

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

[45] Stefan Krastanov and Liang Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific reports*, 7(1):11003, 2017.

[46] Savvas Varsamopoulos, Koen Bertels, and Carmen Garcia Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 69(2):300–311, 2019.

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

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

[49] Arshpreet Singh Maan and Alexandru Paler. Machine learning message-passing for the scalable decoding of qldpc codes. *npj Quantum Information*, 11(1):78, 2025.

[50] Ameya S Bhave, Navnil Choudhury, and Kanad Basu. Hypernq: A hypergraph neural network decoder for quantum ldpc codes. *arXiv preprint arXiv:2511.01741*, 2025.

[51] Hanyan Cao, Feng Pan, Yijia Wang, and Pan Zhang. qecgpt: decoding quantum error-correcting codes with generative pre-trained transformers. *arXiv preprint arXiv:2307.09025*, 2023.

[52] Xiangjun Mi and Frank Mueller. Toward uncertainty-aware and generalizable neural decoding for quantum ldpc codes. *arXiv preprint arXiv:2510.06257*, 2025.

[53] Yue Wu and Lin Zhong. Fusion blossom: Fast mwpm decoders for qec. In *2023 IEEE international conference on quantum computing and engineering (QCE)*, volume 1, pages 928–938. IEEE, 2023.

[54] Namitha Liyanage, Yue Wu, Alexander Deters, and Lin Zhong. Scalable quantum error correction for surface codes using fpga. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 916–927. IEEE, 2023.

[55] Ge Yan, Shanchuan Li, and Yuxuan Du. Rethink the role of neural decoders in quantum error correction. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

[56] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[57] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[58] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[59] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863*, 2019.

[60] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737, 2018.

[61] Edwin Vargas, Claudia V Correa, Carlos Hinojosa, and Henry Arguello. Biper: Binary neural networks using a periodic function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5684–5693, 2024.

[62] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. Neo-qec: Neural network enhanced online superconducting decoder for surface codes. *arXiv preprint arXiv:2208.05758*, 2022.

[63] Oscar Higgott. PyMatching: A python/c++ library for decoding quantum error correcting codes with minimum-weight perfect matching. `https://github.com/oscarhiggott/PyMatching`, 2021. Accessed: 2026-05-07.

[64] Joschka Roffe. BP+OSD: Belief propagation with ordered statistics post-processing for decoding quantum ldpc codes. `https://github.com/quantumgizmos/bp_osd`, 2020. Accessed: 2026-05-07.

[65] Chang Sun, Zhiqiang Que, Thea Aarrestad, Vladimir Loncar, Jennifer Ngadiuba, Wayne Luk, and Maria Spiropulu. HGQ: High granularity quantization for real-time neural networks on fpgas. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 79–91, 2026.

A Extended preliminaries

A.1 Detailed definitions of QEC codes and memory experiments

Stabilizer formalism and code matrices. QEC protects quantum information by redundantly encoding k logical qubits into an entangled state of n physical qubits, denoted by the parameters $[[n, k, d]]$, where d represents the code distance. Let $\mathcal{P}_n$ be the n-qubit Pauli group. The code space is strictly defined as the simultaneous $+1$ eigenspace of an Abelian stabilizer group $\mathcal{S} = \langle S_1, \ldots, S_{n-k} \rangle \subset \mathcal{P}_n$. For Calderbank-Shor-Steane (CSS) codes—the foundational class of codes evaluated in this work—the stabilizer generators can be exclusively partitioned into X-type and Z-type operators. This decoupling allows the quantum error correction condition to be mapped to dual classical linear codes, governed by the parity-check matrices H_X and $H_Z \in \mathbb{F}_2^{(n-k)/2 \times n}$. When a physical error e occurs, the hardware measures the stabilizers to extract a deterministic binary syndrome vector $s = He \pmod 2$. Analogous to classical coding theory, the unmeasured logical degrees of freedom are defined by the generator matrix G (with sub-matrices G_X and G_Z), which satisfies the orthogonality condition $HG^T = 0 \pmod 2$. The rows of the generator matrix G explicitly dictate the support of the non-trivial logical operators $\bar{X}$ and $\bar{Z}$ that commute with all stabilizers but alter the encoded logical state.

Planar surface codes: open boundaries. Planar surface codes are topologically defined on a 2D square lattice with open spatial boundaries. Because the lattice is truncated, it enforces strict boundary conditions, which dictate the termination points for the X- and Z-stabilizers. This planar truncation naturally limits the code capacity, strictly encoding $k = 1$ logical qubit regardless of the lattice size. The parity-check matrix H_{surf} is dynamically constructed by placing data qubits on the edges and stabilizers on the vertices/faces, governed by the local polynomial $P_{\text{surf}} = 1 + x + y + xy$. Due to the open boundaries, translational symmetry is broken at the edges. Consequently, the generator matrix G_{surf} dictates that the logical operators must form continuous string-like Pauli chains that physically span across the entire lattice, connecting opposing boundaries of the same type.

BB codes: toric topology. In stark contrast to planar codes, BB codes belong to a family of quantum low-density parity-check (qLDPC) codes defined on a periodic quasi-cyclic torus. This topology fundamentally lacks spatial boundaries, perfectly preserving structural symmetry across the entire graph. Defined by spatial dimensions l and m, the translation operators strictly obey the cyclic moduli $x^l - 1 = 0$ and $y^m - 1 = 0$, yielding $n = 2lm$ data qubits. The parity-check matrices for BB codes are explicitly formulated as block matrices $H_X = [A, B]$ and $H_Z = [B^T, A^T]$. Here, A and B are drawn from the polynomial ring $\mathbb{F}_2[x, y]/\langle x^l - 1, y^m - 1 \rangle$, specifically $A(x, y) = x^3 + y + y^2$ and $B(x, y) = x + x^2 + y^3$. This advanced algebraic design generates a highly connected expander topology. Crucially, the absence of boundaries and the complex syndrome routing allow a single BB code block to encode $k = 12$ logical qubits (e.g., the standard $[[144, 12, 12]]$ configuration), drastically improving the hardware encoding rate compared to standard surface codes. Their generator matrices G_{BB} yield logical operators that wrap around the periodic cycles of the torus.

Memory experiments. While universal fault-tolerant quantum computation ultimately requires dynamic routing and logic gates (e.g., via lattice surgery or code deformation), the fundamental prerequisite for evaluating hardware or decoder performance is the logical memory experiment. In this work, we focus exclusively on memory experiments, which aim to perfectly preserve a static logical state over consecutive QEC cycles. Specifically, in a Z-memory experiment, the system is initialized in the logical $|0\rangle$ state. Over T sequential rounds of syndrome extraction, the decoder is solely tasked with tracking physical X-type errors (bit-flips) that trigger the Z-stabilizers. Because no logical gates are applied to intentionally manipulate the state, the decoder's objective simplifies strictly to inferring the parity of the physical errors overlapping with the canonical support $\Omega(\bar{Z}_j)$. If the decoder successfully prevents the accumulation of uncorrected errors from forming a logical $\bar{X}$ chain, the memory is preserved.

A.2 Circuit-level noise modeling

To evaluate fault-tolerant decoders accurately without direct access to quantum hardware, rigorous simulation of the physical noise environment is paramount. Unlike idealized phenomenological

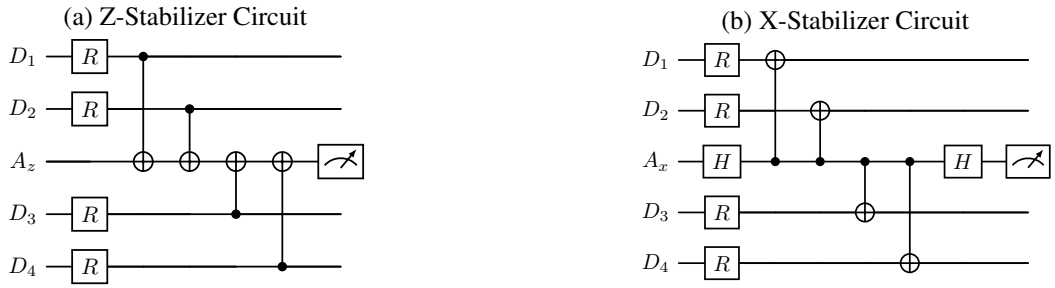


Figure 5: **Standard syndrome extraction circuits and the origin of circuit-level noise.** This figure illustrates the canonical circuit-level implementation for weight-4 stabilizers. (a) The Z-stabilizer measures the parity $Z_1Z_2Z_3Z_4$ by sequentially entangling the data qubits (D_1 to D_4) with the ancilla (A_z). (b) The X-stabilizer measures the parity $X_1X_2X_3X_4$, utilizing Hadamard gates (H) on the ancilla to facilitate measurement in the X-basis. Crucially, the entangling operations (CNOTs) are executed in a strictly sequential schedule. This temporal staggering implies that an unexpected physical fault occurring during an intermediate gate (e.g., an error on R) can propagate through subsequent CNOTs, cascading into multi-qubit correlated errors (hook errors) before the final measurement is triggered.

models where syndromes are assumed to be extracted instantaneously, realistic QEC routines rely on a sequence of noisy two-qubit entangling gates between data and measure qubits (as illustrated in Fig. 5). Because these operations are executed sequentially in a specific schedule, intermediate physical faults can propagate and cascade into multi-qubit correlated errors (e.g., hook errors) during the syndrome extraction circuit. Capturing these complex spatiotemporal error dynamics necessitates a robust circuit-level noise model.

Fundamentally, QEC decoding is a maximum likelihood inference problem aimed at deducing the most probable logical equivalence class given an observed syndrome. The underlying physical error mechanisms that generate these syndromes are formalized as a detector error model (DEM). A DEM maps physical circuit-level faults to the prior probabilities of specific detector events and their corresponding logical observables. Because a trained neural decoder implicitly overfits to the error priors of its training DEM, its generalization capability across different noise landscapes is a critical metric for practical deployment. In this work, we employ the Stim framework to simulate and extract diverse DEMs, comprehensively demonstrating the generalization robustness of the proposed PP decoder. To approximate complex physical error mechanisms, Stim models circuit-level noise by inserting specific probabilistic error channels into the scheduled circuit operations, primarily utilizing four distinct instructions: `after_clifford_depolarization`, `after_reset_flip_probability`, `before_round_data_depolarization`, and `before_measure_flip_probability`. Different DEMs are thus constructed by varying the base probability distributions across these four channels.

Specifically, we investigate three distinct DEMs representing different levels of hardware realism:
1. Standard Uniform Depolarizing Noise (Uniform): This is the most widely adopted theoretical baseline for circuit-level noise. In this DEM, all four aforementioned error channels (gate depolarizing, initialization, idling, and measurement errors) are assigned an identical, independent probability p. While analytically straightforward, the uniform model fails to capture the intrinsic asymmetric error profiles prevalent in actual quantum devices.
2. Superconducting-Inspired Noise (SI1000): To bridge the gap between theoretical uniformity and hardware reality, Google proposed the SI1000 model [38]. This DEM reflects the highly asymmetric error profiles and typical operation durations of modern superconducting architectures, assuming a 1000 ns syndrome extraction cycle. Given a base physical error rate p_{base}, the specific probabilities are empirically scaled as follows: `after_clifford_depolarization` is set to p_{base}, `after_reset_flip_probability` to $2p_{\text{base}}$, `before_round_data_depolarization` to $2p_{\text{base}}$, and `before_measure_flip_probability` to $5p_{\text{base}}$. Recent state-of-the-art neural decoders heavily rely on SI1000 as a standard evaluation benchmark.
3. Hardware-Calibrated Sycamore Noise: Beyond simulated environments, we also evaluate the PP decoder against a DEM directly calibrated from an existing quantum hardware platform—the Google Sycamore processor with surface code. By fitting empirical correlation data extracted from actual memory experiment measurements to the detector error model, this configuration provides a high-

Table 1: **Temporal breakdown of a standard superconducting QEC cycle.** Data is adapted based on the SI1000 model for transmon architectures [38]. The hard $1\mu s$ real-time constraint is fundamentally bottlenecked by the measurement and reset operations.

Layer	Operation	Duration (ns)	Cumulative (ns)
1	Qubit Reset / Initialization	200	200
2	Single-qubit Gates (H, etc.)	25	225
3–6	Two-qubit Gates (CNOT/CZ)	4×30	345
7	Measurement Tone	600	945
8	Photon Depletion / Idling	55	**1000**

fidelity empirical representation of the complex, correlated noise landscape intrinsic to state-of-the-art superconducting processors, serving as the ultimate test for real-world deployability.

It is important to note that while the circuit-level noise configurations (e.g., the specific gate error rates defined in SI1000 or uniform) are agnostic to the underlying quantum code, the resulting DEMs are strictly code-dependent. In our evaluations, we apply these three universal noise configurations to the specific syndrome extraction circuits of the planar surface codes and the uniform configuration to BB codes, utilizing Stim to compile the respective code-specific DEMs for decoder training and inference.

A.3 Decoding latency

The imposition of a strict sub-microsecond decoding latency is not an arbitrary algorithmic target, but a fundamental hardware constraint dictated by the operational characteristics of transmon-based superconducting processors. In standard topological QEC protocols, the decoder must finish inferring the error configuration and calculating the logical parity before the quantum state completely decoheres, or before the measurement buffer is overwritten by the subsequent syndrome extraction round.

For superconducting qubits, this timing window is notoriously tight. As formalized by the SI1000 noise model [38], a complete QEC cycle (t_{cycle}) is heavily dominated by the sluggish duration of cavity-based readout and photon depletion, rather than the ultra-fast unitary gates. As broken down in Table 1, while single- and two-qubit gates are executed within tens of nanoseconds, the measurement and initialization phases consume the vast majority of the cycle, bounding the total latency to approximately $1\mu s$. Consequently, a practical neural decoder must physically execute its entire forward pass within this $1\mu s$ margin to support real-time active feed-forward, presenting the ultimate engineering bottleneck that our PP architecture addresses.

Beyond superconducting circuits, the stringency of this latency requirement varies drastically depending on the underlying physical modalities, largely dictated by their intrinsic coherence times (T_1 and T_2). For instance, trapped-ion and neutral-atom architectures benefit from exceptionally long coherence times, often spanning seconds to minutes. Although their entangling gate fidelities are competitive, the physical operations—such as atomic shuttling, laser cooling, and Rydberg excitations—are orders of magnitude slower than microwave pulses, expanding a single QEC cycle to the millisecond regime ($\sim 1 - 10$ ms). In these platforms, the decoding latency constraint is inherently relaxed, allowing classical decoders executing on standard CPUs ample time to process the syndromes. Therefore, by aggressively optimizing our neural decoder to satisfy the extreme $1\mu s$ threshold of superconducting systems, the proposed architecture trivially guarantees real-time deployment compatibility across all other major quantum hardware modalities with slower clock cycles.

A.4 Basic concepts of FPGAs

Unlike general-purpose processors (CPUs or GPUs) that possess fixed computational datapaths and rigid instruction sets, FPGAs provide a distributed array of reconfigurable logic elements and routing interconnects that can be synthesized into custom digital circuits. The hardware mapping of any neural network architecture onto an FPGA is fundamentally constrained by three primary physical resource pools: block RAM (BRAM), look-up tables (LUTs), and digital signal processors (DSPs).

BRAM for on-chip storage. BRAMs are discrete, high-speed static RAM blocks distributed throughout the FPGA fabric. They serve as high-bandwidth local caches with single-cycle latency access. To avoid the severe latency penalty of off-chip DRAM access, a low-latency neural decoder typically requires its parameters to fit entirely within the aggregate BRAM capacity of the device (approximately $30 \sim 100$ MB on modern high-end FPGAs).

LUTs for logic primitives. LUTs are the finest-grained configurable units on an FPGA. A standard LUT (e.g., in AMD Versal and UltraScale devices) is essentially a small memory array configured to implement any arbitrary Boolean function of up to 6 inputs. They are overwhelmingly abundant, with typical high-end devices hosting millions ($\sim 10^6$ to 10^7) of LUTs. While fundamentally designed for control logic and bitwise operations, LUTs can be chained together via dedicated carry-logic to construct arithmetic operators (such as adders and multipliers). However, the hardware cost of synthesizing arithmetic from pure logic primitives scales non-linearly with precision. For instance, the logic complexity (and thus LUT consumption) of an N-bit multiplier scales approximately as $\mathcal{O}(N^2)$.Consequently, utilizing LUT clusters for ultra-low precision arithmetic (e.g., INT4 or binary operations) is highly efficient. Conversely, scaling to standard quantized precisions like INT8 or FP16 drastically bloats the LUT footprint and exacerbates routing congestion, causing severe timing degradation. Therefore, INT4 generally represents the practical upper bound for massive, purely LUT-based arithmetic arrays.

DSPs for arithmetic engines. In stark contrast to LUTs, DSP slices are specialized, hard-wired silicon blocks engineered exclusively for high-speed, high-precision arithmetic. A typical DSP block contains a pre-configured wide multiplier tightly coupled with a wide accumulator (e.g., a 27×24-bit multiplier and a 58-bit accumulator in the DSP58 slices of AMD Versal devices). Because they are physical ASICs embedded within the programmable fabric, they operate at significantly higher clock frequencies than LUT-based arithmetic. However, DSPs are a relatively scarce resource, typically numbering only on the order of 10^3–10^4 per high-end device (e.g., 6,864 DSP58 slices on the VP1902 used in this work). In standard hardware-aware neural network deployments, one Multiply-Accumulate (MAC) operation is logically mapped to one physical DSP slice. A critical inefficiency arises when evaluating low-precision models (e.g., INT8) on these blocks. While a DSP can mathematically execute an 8×8-bit multiplication, feeding 8-bit inputs into a hardware unit physically wired for 27×24-bit arithmetic leaves the vast majority of its internal logic gates idle. Despite this massive under-utilization of internal capacity, the entire DSP slice is consumed. Because DSPs cannot be arbitrarily partitioned to execute multiple independent INT8 MACs concurrently, relying on DSPs for low-precision operations fundamentally throttles the maximum achievable parallelism to the strict, scarce physical count of the DSP blocks.

B Related works

B.1 QEC codes and conventional decoders

In fault-tolerant quantum error correction, the underlying structure of a stabilizer code fundamentally dictates its syndrome patterns and optimal decoding strategy. Surface codes and related topological families (e.g., color codes [39]) rely on geometrically local stabilizer checks across a 2D lattice. Because their error correlations are highly localized, decoding is typically handled by matching and local graph algorithms operating on structured 2D or 3D spacetime graphs [40, 5]. Conversely, qLDPC codes [41, 8] achieve higher encoding efficiency through sparse, non-local connections. Their irregular Tanner graphs require iterative message-passing methods like belief propagation, which must navigate inherent challenges such as degeneracy and short cycles [18]. Regardless of the specific code family, practical circuit-level noise demands repeated syndrome extraction. This introduces a temporal dimension, transforming the decoding process into a complex spatiotemporal inference task. Consequently, the decoding methodologies reviewed in the following sections are categorized according to these distinct graph architectures and their associated spatiotemporal dynamics.

Traditional classical decoders for stabilizer codes are broadly categorized by the distinct graph architectures induced by their underlying code families. For planar surface codes, decoding is conventionally formulated as a graph-based optimization problem over the spacetime syndrome volume. The ubiquitous baseline is Minimum-Weight Perfect Matching (MWPM), which resolves detection events by mapping them to a matching problem solvable via Blossom-type algorithms [42, 5]. Modern implementations of MWPM have aggressively optimized execution overhead at scale by exploiting

topological sparsity and streaming-style dataflows [35]. Alternatively, Union-Find decoders [43] operate on the same local lattice structure but replace global optimization with greedy, localized cluster growth and disjoint-set merging, achieving near-linear-time empirical complexity [43]. Comprehensive discussions covering MWPM, Union-Find, and related local graph-based methodologies are widely available in recent literature [40].

Conversely, for code families defined by highly non-local, irregular Tanner graphs—such as quantum LDPC (qLDPC) codes—iterative message-passing algorithms like Belief Propagation (BP) serve as the natural decoding framework, drawing heavily from classical telecommunications. However, transitioning classical BP to the quantum domain introduces fundamental challenges; the highly degenerate nature of quantum errors and the inevitable presence of short trapping sets severely degrade native convergence and inference accuracy [17]. To mitigate this, augmented variants rely on complex algebraic post-processing. Most notably, BP combined with Ordered-Statistics Decoding (BP+OSD) significantly improves algorithmic robustness by resolving lingering parity ambiguities via Gaussian elimination. While this incurs substantial computational overhead, BP+OSD has firmly established itself as the de facto accuracy baseline for evaluating qLDPC codes [44].

B.2 Neural decoders

For topological codes characterized by geometrically regular spacetime syndrome volumes, a diverse spectrum of neural architectures has been investigated. Early learning-based approaches deployed multi-layer perceptrons (MLPs) on flattened syndrome vectors [45, 22]; however, these densely parameterized models quickly encounter severe scalability bottlenecks as the code distance increases. To systematically exploit the inherent spatial locality and translation invariance of topological codes, convolutional neural networks (CNNs) were subsequently introduced, natively processing continuous syndrome histories as 2D or 3D spatial volumes [46]. Furthermore, because realistic circuit-level noise unfolds dynamically over repeated measurement rounds, sequence modeling techniques—such as recurrent neural networks (RNNs) and temporal convolutional networks (TCNs)—have been adopted to track complex temporal error correlations and measurement faults [23, 30]. Most recently, Transformer-based architectures have redefined the state-of-the-art for neural decoding. By leveraging global attention mechanisms, Transformers can elegantly resolve long-range error chains and natively integrate analog soft-measurement information, achieving unprecedented logical accuracy on experimental hardware [10, 11]. To reduce the training cost of such large models, Yan et al. [47] propose neural transfer unification, which aligns decoding tasks across code distances so that Transformer decoders trained on small codes transfer to larger ones, scaling to surface codes of distance up to 25.

Conversely, for code families defined by irregular parity-check connectivity—such as qLDPC codes—neural decoding paradigms must operate directly on the underlying Tanner or factor graphs. A prominent algorithmic direction is Neural Belief Propagation (NBP), which explicitly unrolls classical BP iterations into a trainable computational graph. By learning parameterized message-update weights end-to-end, NBP effectively dampens the harmful oscillations caused by short cycles and code degeneracy that traditionally plague classical message-passing [48]. Beyond strict BP-style updates, more expressive Graph Neural Networks (GNNs) have been deployed to learn generalized latent representations, capturing higher-order structural correlations within the parity matrix [49]. The landscape of neural decoding is also rapidly expanding to encompass novel representational frameworks. Recent innovations include hypergraph networks that natively encode complex multi-body stabilizer constraints [50], sequence-to-sequence generative models that frame the syndrome-to-logical mapping as a language translation task [51], and uncertainty-aware probabilistic decoders designed to output calibrated confidence estimates alongside parity predictions [52]. Nevertheless, despite these algorithmic triumphs, deploying such highly parameterized neural architectures under the strict real-time execution constraints of physical quantum processors remains a formidable open challenge.

B.3 Real-time QEC decoding

Real-time decoding is a critical requirement for fault-tolerant quantum computation, as slow decoders cause an exponential backlog of errors, negating the lifetime advantage of logical qubits. Despite notable progress, achieving true real-time decoding under the stringent latency, throughput, and resource constraints of practical quantum systems remains a significant challenge, with existing solutions each facing fundamental limitations.

At the algorithmic level, Fusion Blossom[53] and Sparse Blossom[35] adapt the blossom algorithm to operate directly on the decoding graph, achieving near-linear time complexity. Fusion Blossom supports large-scale parallel decoding through a recursive divide-and-conquer strategy, but its throughput relies on substantial general-purpose computing resources, which may become prohibitively expensive in large-scale quantum data center scenarios.

In terms of hardware acceleration, Helios[54] implements a fully distributed Union-Find decoder on FPGA, exploiting vertex-level parallelism to significantly reduce decoding latency. However, the number of processing elements grows as $O(d^3)$, causing logic resource consumption to expand rapidly with code distance, thereby limiting the maximum code distance that a single chip can support. Micro Blossom adopts a heterogeneous CPU-FPGA architecture and achieves the first sub-microsecond exact MWPM decoding, but its worst-case latency can degrade significantly under complex error patterns, posing a potential risk for hard real-time systems.

In neural network decoding, AlphaQubit [10] achieves accuracy surpassing traditional decoders on experimental data through a two-stage training strategy. Building upon this, AlphaQubit 2 [11] utilizes curriculum learning to significantly expand the decodable code distance up to $d = 23$. However, a fundamental discrepancy exists between their reported performance and true real-time hardware execution. Both architectures intrinsically prioritize throughput over strict latency. By processing a massive temporal window of syndrome rounds as a single batch, they report an amortized "per-round latency" that mathematically appears extremely low. Unfortunately, this batched metric physically masks the massive end-to-end execution delay. In practical fault-tolerant protocols that require rapid mid-circuit feedback—such as updating the Pauli frame prior to a transversal T gate execution—the quantum processor must wait for the entire batch to finish processing, incurring millisecond-level idle times. Consequently, despite their impressive algorithmic accuracy, these throughput-optimized architectures remain fundamentally misaligned with the strict 1 μs real-time turnaround demanded by active superconducting control systems.

Closest to our work, Yan et al. [55] revisit neural decoders from a deployment perspective, benchmarking five architectural paradigms (MLP, CNN, TCN, Transformer, and GNN) on surface codes up to $d = 9$ under explicit accuracy–latency constraints, and compressing them via quantization and pruning for FPGA deployment. They find that accuracy gains are driven more by training-data scale than by architectural choice. PP follows this deployment-oriented perspective and further co-designs the precision allocation with the decoding pipeline.

Overall, existing decoding solutions still struggle to simultaneously balance accuracy, speed, and resource overhead—how to meet real-time requirements while maintaining high decoding accuracy remains a core open problem.

B.4 Binary neural networks

Binary neural networks (BNNs) represent the extreme case of network quantization [56], constraining both weights and activations to a single bit (+1/-1), thereby replacing floating-point multiply-accumulate operations with bitwise XNOR [57] and bit-count operations, achieving computational speedup and memory compression. Early work by BinaryNet[58] established the binarized training framework, approximating the gradient of the sign function via the straight-through estimator. Since direct binarization introduces severe quantization error, XNOR-Net[57] proposed using analytically computed scaling factors to re-weight the output of binary convolutions, substantially improving accuracy on ImageNet. XNOR-Net++[59] further fused the scaling factors into learnable parameters. For gradient approximation, Bi-Real Net[60] adopted a piecewise polynomial function in place of STE, combined with real-valued skip connections to enhance representational capacity. BiPer[61] introduced a square-wave periodic function to replace the sign function for binarization, with a sinusoidal function serving as the gradient surrogate, flexibly controlling the trade-off between quantization error and gradient stability through a frequency parameter.

Despite such progress in conventional domains, work combining BNNs with QEC remains scarce. To our knowledge, NEO-QEC[62] is the only prior effort in this direction. NEO-QEC integrates an XNOR-Net-based binarized CNN with a superconducting digital circuit decoder, and demonstrates the feasibility of using binary neural networks for online surface code decoding under the low-temperature, low-power constraints of superconducting qubits. However, its evaluation is limited to a phenomenological noise model, and its decoding accuracy remains significantly below that of

global minimum-weight perfect matching. Moreover, its hardware resource estimation was performed before detailed routing, so the actual implementation cost remains to be determined.

Compared to binarized networks, LATTE [36] steps back in quantization precision and adopts INT8, embedding a lightweight neural network into FPGA-based quantum control hardware for local syndrome preprocessing. Through a hybrid architecture with streaming asynchronous block decoding on CPU, it reduces transmission bandwidth while maintaining acceptable accuracy. However, its local decoding unit still relies on INT8 multiplication and thus falls short of the gate-level efficiency of binary networks; its global decoding component still depends on a conventional MWPM decoder, rather than a fully neural decoding scheme; moreover, as the code distance increases, the qubit region covered by a single FPGA becomes insufficient, necessitating distributed multi-board deployment, and the impact of cross-board communication latency and synchronization overhead at larger scales remains unclear.

C Model implementation details

Surface-code decoder. The surface-code decoder takes a flattened detector-event vector $\mathbf{s} \in \{0,1\}^{N_{\mathrm{det}}}$ as input. We denote the batch size by B, the detector time-slice count by T_{det}, the feature width by C_{feat}, and the stabilizer-grid size by $G_y \times G_x$. In all surface-code experiments, $C_{\mathrm{feat}} = 128$ and the number of syndrome-extraction rounds is set to $r = d$. Here T_{det} counts Stim detector-coordinate time slices, including boundary slices, and is therefore distinct from the round count r. For the rotated memory-Z circuits generated by Stim under this setting, the detector geometry satisfies

$$T_{\mathrm{det}} = G_y = G_x = d + 1, \qquad M_Z = M_X = \frac{d^2 - 1}{2}, \qquad N_{\mathrm{det}} = d(d^2 - 1),$$

where M_Z and M_X denote the number of Z- and X-type stabilizer tokens per detector time slice. Detector coordinates from the Stim circuit are used only to construct this tokenization and the subsequent grid placement; no distance-specific trainable parameters are introduced. For example, at $d = 9$ this gives $N_{\mathrm{det}} = 720$, $T_{\mathrm{det}} = G_y = G_x = 10$, and $M_Z = M_X = 40$. The token layout is padded: boundary-time slots that do not correspond to a Stim detector are filled with zero detector bits before lookup, so $T_{\mathrm{det}}(M_Z + M_X)$ need not equal N_{det}.

The embedding layer maps each stabilizer token through three discrete local-pattern indices. The same-type spatial index uses the current detector bit and its four diagonal same-type neighbors, the temporal index uses the previous-round and current detector bits at the same site, and the cross-type hint index uses four nearby opposite-type detectors. These indices are embedded with lookup tables of size 32, 4, and 16, respectively. The three embeddings are summed with a sinusoidal time encoding and refined by a lightweight residual stem MLP. The resulting Z- and X-token features are scattered to a common stabilizer grid,

$$(\mathbf{E}_Z, \mathbf{E}_X) \mapsto \mathbf{U} \in \mathbb{R}^{B \times C_{\mathrm{feat}} \times T_{\mathrm{det}} \times G_y \times G_x}.$$

The perception backbone follows a temporal-spatial-temporal layout,

$$\mathbf{V} = \mathrm{TCN}_2 \circ \mathrm{SRes}_4 \circ \mathrm{SRes}_2 \circ \mathrm{SRes}_1 \circ \mathrm{TCN}_1(\mathbf{U}).$$

Each TCN block applies two causal Conv1D layers along the detector-time axis independently at every grid site, with kernel size 3 and dilations $(1, 3)$. The three spatial residual blocks are applied independently at each detector time slice, and each block consists of two 3×3 Conv2D layers. For SRes_ρ with $\rho \in \{1, 2, 4\}$, the first Conv2D uses spatial dilation ρ, and the second uses dilation 1.

The readout first applies a local 2×2 Conv2D to project detector-grid features onto the $d \times d$ data-qubit grid. For the memory-Z task, the rotated surface code has d equivalent minimal representatives of the target logical string. The decoder therefore averages data-qubit features over the column dimension for each row string, applies a shared row head to each pooled feature, and averages the resulting d row logits. This produces one logit per detector time slice, i.e., an output in $\mathbb{R}^{B \times T_{\mathrm{det}}}$. Table 2 summarizes the complete architecture and deployed precision. We use Wb_{w}Ab_{a} to denote b_{w}-bit weights and b_{a}-bit input activations; thus W1A32 denotes binary weights with FP32 input activations, whereas FP32 denotes full-precision weights and activations, W1 lookup denotes binary embedding tables without activation multiplication.

Table 2: Layer-wise architecture and deployed inference precision of the surface-code decoder. The feature width is fixed to $C_{\text{feat}} = 128$.

Stage	Layer	Shape	Precision
Local embedding	Pattern lookups, table sizes $32/4/16$, plus time encoding	$B \times N_{\text{det}} \to B \times T_{\text{det}} \times M_* \times C_{\text{feat}}$, $* \in \{Z, X\}$	W1 lookup
	Residual stem MLP	$C_{\text{feat}} \to 2C_{\text{feat}} \to C_{\text{feat}}$	W1A2
Grid scatter	Z/X tokens to stabilizer grid	$B \times T_{\text{det}} \times M_* \times C_{\text{feat}} \to$ $B \times C_{\text{feat}} \times T_{\text{det}} \times G_y \times G_x$	–
Backbone	TCN_1: causal Conv1D $\times 2$, $k = 3$, dil. $(1, 3)$ SRes_1: Conv2D $\times 2$, $k = 3$, dil. $(1, 1)$ SRes_2: Conv2D $\times 2$, $k = 3$, dil. $(2, 1)$ SRes_4: Conv2D $\times 2$, $k = 3$, dil. $(4, 1)$ TCN_2: causal Conv1D $\times 2$, $k = 3$, dil. $(1, 3)$	$B \times C_{\text{feat}} \times$ $T_{\text{det}} \times G_y \times$ G_x	W1A2
Readout	Data projection: 2×2 Conv2D	$B \times C_{\text{feat}} \times T_{\text{det}} \times G_y \times G_x \to$ $B \times C_{\text{feat}} \times T_{\text{det}} \times d \times d$	W1A4
	Column pooling per row + shared head + row averaging	$B \times C_{\text{feat}} \times T_{\text{det}} \times d \times d \to B \times T_{\text{det}} \times$ $d \times C_{\text{feat}} \to B \times T_{\text{det}} \times d \to B \times T_{\text{det}}$	W1A4 W1A32 final

BB-code decoder. The BB-code decoder uses the same flattened detector-event input $\mathbf{s} \in \{0, 1\}^{N_{\text{det}}}$ and the same notation B, T_{det}, and C_{feat} as above. The BB Tanner graph is represented on a periodic $\ell \times m$ lattice with two check types, $n = 2\ell m$ data qubits, and k logical qubits. For the BB $[[72, 12, 6]]$ code used in our experiments, $\ell = m = 6$, $n = 72$, $k = 12$, $T_{\text{det}} = 7$, and $N_{\text{det}} = 432$. The mapper arranges detector events into Z- and X-check tokens on the periodic lattice, producing a tensor layout indexed by $T_{\text{det}} \times 2 \times \ell \times m$. Some boundary-time check-token slots are absent and are masked by validity embeddings; therefore the token layout has $T_{\text{det}} \times 2 \times \ell m$ slots, while only $N_{\text{det}} = 432$ detector events are present.

For each check token, the embedding module constructs a same-type spatial pattern from the current detector bit and its K_{same} polynomial same-type neighbors, a temporal pattern from the previous-round and current detector bits at the same check site, and a cross-type hint pattern from K_{cross} nearby opposite-type checks. For the BB $[[72, 12, 6]]$ instance, $K_{\text{same}} = 12$ and $K_{\text{cross}} = 9$. The same-type spatial pattern is augmented with all pairwise products and projected to C_{feat} dimensions, while the temporal, cross-type, check-type, and validity lookup tables have 4, $2^{K_{\text{cross}}}$, 2, and 2 entries, respectively. These features are summed with a sinusoidal time encoding and refined by a residual stem MLP, giving

$$\mathbf{E} \in \mathbb{R}^{B \times T_{\text{det}} \times 2 \times \ell \times m \times C_{\text{feat}}}.$$

The backbone follows the same temporal-spatial-temporal organization as the surface-code decoder, but replaces planar Conv2D layers with polynomial graph residual blocks:

$$\mathbf{H} = \text{TCN}_2 \circ \text{PolyGraphResBlock}_3 \circ \text{PolyGraphResBlock}_2 \circ \text{PolyGraphResBlock}_1 \circ \text{TCN}_1(\mathbf{E}).$$

Each TCN block is a residual block containing two causal Conv1D layers along the detector-time axis, both with kernel size 3 and dilations $(1, 1)$. Each polynomial graph residual block contains two polynomial graph convolution layers. A polynomial graph convolution linearly mixes the ordered same-type neighbor features and the ordered cross-type neighbor features, concatenates them with the center token feature, and projects the resulting $3C_{\text{feat}}$-dimensional feature back to C_{feat} dimensions.

After the check-token backbone, a check-to-data projection maps stabilizer-check features to physical data-qubit features. For each data qubit, the model averages connected Z-check and X-check features using the parity-check matrices H_Z and H_X, concatenates these two averaged features with a learned data-qubit embedding, and applies a projection MLP from $3C_{\text{feat}}$ to C_{feat}. The logical readout uses an optimized weight-6 logical basis with $k = 12$ representatives. It mean-pools data-qubit features over each support and applies an independent logical head $\text{Linear}(C_{\text{feat}}, H_r) \to \text{GELU} \to \text{Linear}(H_r, 1)$ with $H_r = 256$. Since this optimized weight-6 basis differs from the default logical basis, the observable labels are transformed into the same basis during training and evaluation. The BB-code

decoder outputs logits in $\mathbb{R}^{B\times T_{\mathrm{det}}\times k}$. Table 3 summarizes the complete architecture and deployed precision.

Table 3: Layer-wise architecture and deployed inference precision of the BB-code decoder. The feature width is fixed to $C_{\mathrm{feat}} = 128$, and the readout hidden width is $H_r = 256$.

Stage	Layer	Shape	Precision
Check-token embedding	Pairwise same-type projection; temporal, cross-type, type, and validity lookups; time encoding	$B\times N_{\mathrm{det}} \to B\times T_{\mathrm{det}}\times 2\times \ell\times m\times C_{\mathrm{feat}}$	W1 lookup
	Residual stem MLP	$C_{\mathrm{feat}} \to 2C_{\mathrm{feat}} \to C_{\mathrm{feat}}$	W1A4
Backbone	TCN_1: causal Conv1D $\times 2$, $k=3$, dil. $(1,1)$ $\mathrm{PolyGraphResBlock}_1$: graph conv $\times 2$ $\mathrm{PolyGraphResBlock}_2$: graph conv $\times 2$ $\mathrm{PolyGraphResBlock}_3$: graph conv $\times 2$ TCN_2: causal Conv1D $\times 2$, $k=3$, dil. $(1,1)$	$B \times T_{\mathrm{det}} \times 2 \times \ell \times m \times C_{\mathrm{feat}}$	W1A4
Check-to-data projection	H_Z/H_X check averaging, data-qubit embedding, projection MLP	$B \times T_{\mathrm{det}} \times 2 \times \ell m \times C_{\mathrm{feat}} \to B \times T_{\mathrm{det}} \times n \times C_{\mathrm{feat}}$	W1 lookup; W1A4
Logical readout	Optimized weight-6 support pooling; $k=12$ independent heads	$B \times T_{\mathrm{det}} \times n \times C_{\mathrm{feat}} \to B \times T_{\mathrm{det}} \times k \times C_{\mathrm{feat}} \to B \times T_{\mathrm{det}} \times k$	W1A4 W1A32 final

C.1 Progressive quantization schedule

Directly converting an FP32 decoder to the final deployed precision caused severe optimization instability in our experiments, especially once both weights and activations were aggressively quantized. We therefore use a progressive quantization schedule: each stage is initialized from the best checkpoint of the preceding stage, and the activation precision is reduced only after the current precision state has recovered decoding accuracy. Tables 4 and 5 list the accepted precision state after each annealing stage. The entries use the same module names as Tables 2 and 3; non-parametric routing and pooling operations are omitted. The final-output column is FP32 only during the initial full-precision pretrain and is W1A32 in all subsequent stages.

C.2 FakeEnd process supervision

Both decoders produce a prediction at every detector-coordinate time slice, but the physical training data provide the true observable only for the complete syndrome history. Training only from this final label gives a sparse supervision signal to early time slices. We therefore use FakeEnd process supervision: a classical decoder is used to generate intermediate pseudo-labels by pretending that the experiment terminates at each detector-coordinate time slice. Here τ indexes detector-coordinate time, which is distinct from both the syndrome-extraction round count r and the metric variable R used below.

Let τ_i be the detector-coordinate time of detector i. For a prefix time τ, we construct a prefix mask $\boldsymbol{\chi}^{(\tau)}$ and a masked detector vector

$$\mathbf{s}^{(\tau)} = \mathbf{s} \odot \boldsymbol{\chi}^{(\tau)}, \qquad \chi_i^{(\tau)} = \mathbf{1}\{\tau_i \le \tau\},$$

which keeps only detector events observed up to time τ and sets all future detector events to zero. A classical decoder $\mathcal{D}_{\mathrm{cl}}$ is then applied to this truncated syndrome,

$$\tilde{\mathbf{y}}^{(\tau)} = \mathcal{D}_{\mathrm{cl}}\left(\mathbf{s}^{(\tau)}\right),$$

giving the FakeEnd label for that prefix. The decoder $\mathcal{D}_{\mathrm{cl}}$ is code dependent rather than architecture dependent: for surface-code experiments we use a matching-based decoder, whereas for BB codes the same construction would use a BP-family decoder. For matching-based generation, when the truncated syndrome ends at an intermediate time, defects are allowed to terminate at the corresponding fake temporal boundary, matching the interpretation that no future measurement information is available.

Table 4: Progressive quantization schedule for the surface-code decoder. Each row is initialized from the previous row; newly reduced precision entries are shown in bold.

Stage	Local embedding	TCN_1	SRes_1	SRes_2	SRes_4	TCN_2	Data projection	Row head	Final
FP32 pretrain	FP32 **W1 lookup**	FP32	FP32	FP32	FP32	FP32	FP32	FP32	FP32
W1A32 warm-up	**W1A32** W1 lookup	**W1A32**	**W1A32**	**W1A32**	**W1A32**	**W1A32**	**W1A32**	**W1A32**	**W1A32**
W1A8 annealing	**W1A8** W1 lookup	**W1A8**	**W1A8**	**W1A8**	**W1A8**	**W1A8**	**W1A8**	**W1A8**	W1A32
W1A4 base	**W1A4** W1 lookup	**W1A4**	**W1A4**	**W1A4**	**W1A4**	**W1A4**	**W1A4**	**W1A4**	W1A32
TCN_2 W1A2	W1A4 W1 lookup	W1A4	W1A4	W1A4	W1A4	**W1A2**	W1A4	W1A4	W1A32
SRes_4 W1A2	W1A4 W1 lookup	W1A4	W1A4	W1A4	**W1A2**	W1A2	W1A4	W1A4	W1A32
SRes_2 W1A2	W1A4 W1 lookup	W1A4	W1A4	**W1A2**	W1A2	W1A2	W1A4	W1A4	W1A32
SRes_1 W1A2	W1A4 W1 lookup	W1A4	**W1A2**	W1A2	W1A2	W1A2	W1A4	W1A4	W1A32
Final deployed	**W1A2**	**W1A2**	W1A2	W1A2	W1A2	W1A2	W1A4	W1A4	W1A32

Table 5: Progressive quantization schedule for the BB-code decoder. Each row is initialized from the previous row; newly reduced precision entries are shown in bold.

Stage	Check-token embedding	TCN_1	PolyGraph $\mathrm{ResBlock}_1$	PolyGraph $\mathrm{ResBlock}_2$	PolyGraph $\mathrm{ResBlock}_3$	TCN_2	Check-to-data projection	Logical readout	Final
FP32 pretrain	FP32 **W1 lookup**	FP32	FP32	FP32	FP32	FP32	FP32 **W1 lookup**	FP32	FP32
W1A32 warm-up	**W1A32** W1 lookup	**W1A32**	**W1A32**	**W1A32**	**W1A32**	**W1A32**	**W1A32** W1 lookup	**W1A32**	**W1A32**
W1A8 annealing	**W1A8** W1 lookup	**W1A8**	**W1A8**	**W1A8**	**W1A8**	**W1A8**	**W1A8** W1 lookup	**W1A8**	W1A32
TCN_2 W1A4 $\mathrm{PolyGraphResBlock}_3$	W1A8 W1 lookup	W1A8	W1A8	W1A8	W1A8	**W1A4**	W1A8 W1 lookup	W1A8	W1A32
W1A4 $\mathrm{PolyGraphResBlock}_2$	W1A8 W1 lookup	W1A8	W1A8	W1A8	**W1A4**	W1A4	W1A8 W1 lookup	W1A8	W1A32
W1A4 $\mathrm{PolyGraphResBlock}_1$	W1A8 W1 lookup	W1A8	W1A8	**W1A4**	W1A4	W1A4	W1A8 W1 lookup	W1A8	W1A32
W1A4	W1A8 W1 lookup	W1A8	**W1A4**	W1A4	W1A4	W1A4	W1A8 W1 lookup	W1A8	W1A32
TCN_1 W1A4	W1A8 W1 lookup	**W1A4**	W1A4	W1A4	W1A4	W1A4	W1A8 W1 lookup	W1A8	W1A32
Final deployed	**W1A4**	W1A4	W1A4	W1A4	W1A4	W1A4	**W1A4**	**W1A4**	W1A32

During training, the model's final-time prediction is supervised by the true observable label $\mathbf{y}$, while the earlier predictions are supervised by the FakeEnd labels. We use the composite objective

$$\mathcal{L} = \mathcal{L}_{\mathrm{final}} + \beta\, \mathcal{L}_{\mathrm{FE}},$$

where $\mathcal{L}_{\mathrm{final}}$ is the binary cross-entropy between the last-time prediction and $\mathbf{y}$, and $\mathcal{L}_{\mathrm{FE}}$ is the average binary cross-entropy between intermediate predictions and $\tilde{\mathbf{y}}^{(\tau)}$. The coefficient β controls the strength of process supervision. This construction is used only during training; evaluation always uses the model prediction from the full detector history.

D Experimental setup

D.1 Evaluation metrics

LER per round. This metric quantifies the likelihood of a logical error happening during a single QEC round. Given the block-level logical error probability $P_{\text{block}}(R)$ measured after R consecutive rounds, we define the logical fidelity as $F(R) = 1 - 2P_{\text{block}}(R)$. Under the assumption that logical errors accumulate independently from round to round, the fidelity follows the exponential form $F(R) = (1-2\epsilon)^R$. Solving this relation yields the per-round logical error rate:

$$\epsilon = \frac{1}{2}\left[1 - (1 - 2P_{\text{block}}(R))^{1/R}\right]. \tag{7}$$

LER per round per qubit. For quantum codes that encode multiple logical qubits, we further define the per-round per-logical-qubit logical error rate. Let k be the number of logical qubits encoded, and $P_{\text{block}}(R)$ the total block-level error probability after R rounds. The effective block-level error probability per logical qubit is

$$P_{\text{LQ}}(R) = 1 - (1 - P_{\text{block}}(R))^{1/k}. \tag{8}$$

To propagate this effective error probability per logical qubit down to a per-round basis, we invert the cumulative relation $F(R) = (1 - 2\epsilon_{\text{LQ}})^R$, where $F(R) = 1 - 2P_{\text{LQ}}(R)$. Solving for ϵ_{LQ} yields:

$$\epsilon_{\text{LQ}} = \frac{1}{2}\left[1 - (1 - 2P_{\text{LQ}}(R))^{1/R}\right]. \tag{9}$$

D.2 Dataset

Online training. For simulated experiments, we generate training samples online using the Stim circuit simulator [34] instead of training from a fixed cached dataset. For each code distance, syndrome-round setting, physical error rate, and circuit-level noise model, we instantiate the corresponding Stim memory circuit and compile its detector sampler. During training, each data-loading worker repeatedly draws fresh batches of detector-event samples $\mathbf{s}$ together with their logical observable labels $\mathbf{y}$ from this sampler. This streaming data pipeline avoids repeatedly revisiting a finite set of rare syndrome histories and reduces the risk that the neural decoder memorizes a cached training set. It also keeps the effective training distribution aligned with the detector error model used for evaluation, since every batch is sampled directly from the same circuit-level noise process.

DEM-derived decoder inputs. For each simulated memory experiment, we also compile the same Stim circuit used for online sampling into a detector error model (DEM). This DEM provides the common classical-decoder input representation used by the non-neural baselines. For surface-code MWPM decoding, the circuit is compiled with `decompose_errors=True`, which decomposes hyperedge error mechanisms into graphlike edges suitable for PyMatching. For the BB-code BP-family baselines, we flatten the DEM and enumerate its elementary error instructions. Each elementary error mechanism defines one column of the binary check matrix H: detector targets populate the corresponding detector-check entries, while logical-observable targets populate the corresponding column of the logical observable matrix L. The probability attached to the DEM instruction is stored as the prior p_e for that column. The resulting tuple (H, L, p_e) is used unchanged for BP+OSD and RelayBP.

Sycamore datasets. We validate our models on experimental data from the Google Sycamore processor, publicly available at `https://zenodo.org/records/6804040`. We select configurations where the number of measurement rounds equals the code distance, specifically $d{=}3, r{=}3$ and $d{=}5, r{=}5$. For $d{=}3$, the dataset provides four center qubit locations; we select center $(7, 5)$ which exhibits the lowest baseline logical error rate among the four options. Using the device-calibrated Stim circuit (`circuit_noisy.stim`) provided with the dataset, we sample 5×10^6 synthetic training examples that reflect the experimentally characterized noise profile. Training hyperparameters match those used for synthetic data. The dataset contains 5×10^4 experimental shots per configuration; we split these into 4.5×10^4 samples for fine-tuning and 5×10^3 samples for testing. Fine-tuning adapts models pretrained on synthetic data to the experimental noise distribution.

Fakeend annealing. To accelerate the decoding-intensive fakeend generation process, we employ a multi-worker, multi-threaded CPU pipeline that generates MWPM pseudo-labels in parallel with the model's GPU training. The mixing weight β that governs the supervisory signal is not held fixed, but follows a controlled annealing schedule: at the beginning of training, β takes a relatively large value, allowing the network to fully learn the intermediate structure of the decoding problem from the dense per-time-step supervision provided by the fakeend labels; as training proceeds, β decays linearly to zero, at which point the objective function effectively reduces to relying solely on the final observable label. This schedule is activated after a warm-up period, ensuring that the network has already established stable intermediate representations before annealing begins. The annealing process endows the model with rich structured guidance during the early phase of training, while enabling it to focus its optimization entirely on directly minimizing the final logical error during the convergence phase, thereby preventing the decoder bias inherent in the intermediate pseudo-labels from exerting a lasting influence on the quality of the final prediction.

D.3 Baseline implementation

We evaluate conventional algorithmic baselines and neural baseline reimplementations under the same Stim-generated memory experiments used by the PP decoder. The neural baselines include a LATTE-style NLDU baseline [36] and a Bi-NEO-QEC pure-BNN baseline inspired by NEO-QEC [62]. For each evaluated code, physical noise rate, and memory circuit, all methods use detector-event samples and logical-observable labels generated from the same Stim circuit-level noise process. Baseline-specific inputs are then constructed from the same detector samples and, when needed, the corresponding detector error model (DEM): MWPM uses a decomposed graphlike DEM, BP-family BB-code decoders use the DEM-derived (H, L, p_e) tuple, and neural baselines embed the same detector events into their own input formats. This ensures that all comparisons share the same circuit-level noise model, detector convention, and logical-observable convention. Unless otherwise stated, decoder hyperparameters are fixed for each baseline and are not tuned per noise point or per sample; the DEM probabilities follow the evaluated physical noise model and are not treated as tunable hyperparameters.

The MWPM baseline is constructed with `PyMatching`. We implement the standard MWPM baseline for surface-code experiments using PyMatching [63]. The graphlike DEM generated with `decompose_errors=True` is loaded directly into PyMatching, and the edge weights are determined from the DEM error probabilities using the package's standard likelihood weighting. Given a detector-event sample $\mathbf{s}$, PyMatching returns the predicted logical observable parity, which is compared against the sampled logical label to estimate the logical error rate.

The BP+OSD baseline is constructed with `bposd`. We implement BP+OSD for BB-code experiments using the open-source `bposd` package [64]. For each fixed BB-code circuit and noise setting, the decoder is initialized once from the DEM-derived parity-check matrix H and prior probabilities p_e, together with the logical observable matrix L. We use product-sum BP with a maximum of 20 iterations, followed by combination-sweep OSD with `osd_method=''osd_cs''` and `osd_order=10`. Given a detector-event sample $\mathbf{s}$, the decoder returns an estimated error vector $\widehat{\mathbf{e}}$ over the DEM error mechanisms. The predicted logical observable is computed as $\widehat{\mathbf{e}}L^T \bmod 2$.

The RelayBP baseline is constructed with `relay-bp`. We implement RelayBP for BB-code experiments using the open-source `relay-bp` package [37], a Rust implementation with Python bindings. RelayBP is evaluated on the same DEM-derived H, L, and p_e as BP+OSD. We use the RelayBP-$2d$ configuration: for the $[[72, 12, 6]]$ BB code, $d = 6$, so both `pre_iter` and `set_max_iter` are set to 12. The remaining parameters are fixed to `alpha=1.0`, `gamma0=0.15`, `gamma_dist_interval=(-0.226,0.622)`, `num_sets=60`, and `stop_nconv=1`. The decoder output is converted to the predicted logical observable using the same logical observable matrix L.

The LATTE-style baseline is constructed with an NLDU and an MWPM residual backend. We replicate the LATTE-style neural local decoding unit (NLDU) baseline for the rotated surface-code memory-Z experiments [36]. The original LATTE system is an FPGA-CPU hybrid decoding architecture, where an FPGA-resident NLDU locally predicts short-range errors and updates the syndrome before a CPU-based block decoding system processes the remaining syndrome stream. Our implementation focuses on the algorithmic NLDU component under our evaluation setup, and does not reproduce LATTE's FPGA streaming pipeline, asynchronous BLDS scheduler, bandwidth measurements, or latency measurements.

For each distance $d \in \{3, 5, 7, 9\}$, we use the same Stim-generated circuit-level memory-Z circuits as the other surface-code experiments, with $R = d$ syndrome-extraction rounds and uniform physical error rate $p = 0.002$. Detector events are embedded into a tensor of shape $B \times 2 \times T \times H \times W$, where the two channels correspond to the two stabilizer-detector types and virtual boundary vertices are marked by the constant value 2. The NLDU follows the lightweight 3D fully convolutional architecture used by LATTE:

$$\mathrm{Conv3D}(2, 7, 3^3) \to \mathrm{Conv3D}(7, 7, 3^3) \to \mathrm{Conv3D}(7, 7, 3^3) \to \mathrm{Conv3D}(7, 6, 1^3),$$

with ReLU activations after the first three convolutional layers. The six output channels represent local predictions for Pauli errors $\{I, X, Y, Z\}$, measurement errors M, and hook errors H.

Training labels are generated from the decomposed Stim detector error model. For each sampled elementary DEM error mechanism, detector targets are mapped to local NLDU output slots according

to their space-time geometry: temporal detector pairs define measurement-error labels, same-time same-type detector pairs define Pauli-error labels, and mixed-type or more complex local components are mapped to hook-error labels. The loss is the sum of a masked cross-entropy loss for the $\{I, X, Y, Z\}$ Pauli channel and masked binary cross-entropy losses for the M and H channels.

During evaluation, the NLDU prediction is converted into a local detector correction and a local logical-parity update. We then decode the residual syndrome using PyMatching MWPM and combine the NLDU logical update with the MWPM residual prediction. A block-level failure is counted when the combined prediction differs from the Stim-sampled observable flip. The resulting block LER is normalized using Eq. (7); hence the LATTE curve shown in the main text reports LER per round, not raw block LER.

The Bi-NEO-QEC baseline is constructed as a pure BNN logical decoder. We also implement a pure binary neural-network baseline inspired by the binarized NEO-QEC decoder [62]. The original NEO-QEC architecture is a two-stage online decoder: a binarized convolutional neural network first predicts local data-qubit and measurement errors on a target syndrome layer, and an online decoder such as QECOOL or QULATIS subsequently decodes the updated syndrome. Our baseline isolates the binary-neural component and evaluates an NN-only logical decoder under the same circuit-level surface-code setting used in our experiments; it does not include the QECOOL/QULATIS second stage and is therefore not a full reproduction of the original NEO-QEC system.

For each code distance, we train a separate flat binary neural network because the number of Stim detectors depends on d. The input is the full detector-event vector $\mathbf{s} \in \{0,1\}^{N_{\mathrm{det}}(d)}$ from one rotated memory-Z experiment, with $N_{\mathrm{det}} = 24, 120, 336, 720$ for $d = 3, 5, 7, 9$, respectively. The network directly outputs one logit corresponding to the logical observable correction bit. Before the first layer, detector bits are mapped from $\{0,1\}$ to $\{-1,+1\}$. The hidden layers use binary weights and binary activations with sign binarization, trained using a clipped straight-through estimator. The deployed architecture is

$$N_{\mathrm{det}} \to 2048 \to 1024 \to 1024 \to 1,$$

where each hidden binary linear layer is followed by batch normalization and a binary activation, and the final binary linear layer outputs a real-valued logit.

The BNN is trained by distilling MWPM decisions generated from the same graphlike Stim detector error model used for the MWPM baseline. Specifically, PyMatching is used only to produce supervised training labels; no matching decoder or other backend decoder is called during validation or final inference. At evaluation time, a distance-specific threshold selected on a validation set converts the output logit into a binary logical correction. A block-level logical failure is counted when this predicted correction bit differs from the observable flip sampled by Stim. The resulting block LER is normalized using Eq. (7); hence the Bi-NEO-QEC curve reports LER per round, not raw block LER.

D.4 Training hyper-parameters

Surface code decoder. Table 6 summarizes the training hyperparameters for the surface-code PP decoder. All precision stages use the same AdamW optimizer with a linear warmup followed by cosine annealing. At each stage transition, the model is initialized from the selected checkpoint of the preceding stage, while the optimizer state and learning-rate schedule are restarted.

BB-code decoder. Table 7 summarizes the training hyperparameters for the BB-code decoder. All progressive stages use AdamW with a linear warmup followed by cosine annealing. At each stage transition, the model is initialized from the selected checkpoint of the preceding stage, while the optimizer state and learning-rate schedule are restarted.

E Extended experimental results

E.1 Ablation study

Quantization precision. We first provide a detailed ablation study to validate the optimal precision configuration of the PP architecture evaluated in our main experiments. As established, our primary

Table 6: Training hyperparameters for the surface-code decoder.

Hyperparameter	Value
Optimizer	AdamW
AdamW betas / eps	$(0.9, 0.999)$ / 10^{-8}
Learning rate	3×10^{-4}
LR schedule	Linear warmup followed by cosine annealing
Warmup steps	1,000
Minimum LR	0.1 times the base learning rate
Batch size	16,384
Training steps	100,000
Validation interval	Every 500 steps
Validation samples	100,000 online samples
Gradient clipping	Global norm 1.0
Weight decay	3×10^{-3}
Early stopping	30
Loss function	BCEWithLogitsLoss

Table 7: Training hyperparameters for the BB-code decoder.

Hyperparameter	Value
Optimizer	AdamW
AdamW betas / eps	$(0.9, 0.999)$ / 10^{-8}
Learning rate	3×10^{-4}
LR schedule	Linear warmup followed by cosine annealing
Warmup steps	1,000
Minimum LR	0.05 times the base learning rate
Batch size	4,096
Training steps	50,000
Validation interval	Every 1,000 steps
Validation samples	100,000 online samples
Weight decay	10^{-2}
Gradient clipping	Global norm 1.0
Early stopping	30
Loss function	BCEWithLogitsLoss

configuration for planar surface codes, PP (A2-A4-A32), successfully achieves a logical error rate that slightly outperforms the highly optimized classical MWPM baseline. Before conducting empirical ablations, we structurally exclude "inverted" precision hierarchies, such as PP (A4-A2-A32). From an information-theoretic perspective, placing a low-precision layer between higher-precision stages is inherently counterproductive; any fine-grained latent features or representational advantages captured by a 4-bit early perception phase would be catastrophically destroyed by the subsequent 2-bit pooling bottleneck. Consequently, our empirical evaluation strictly focuses on monotonically escalating or flat configurations to rigorously justify the necessity of the A2 $\rightarrow$ A4 $\rightarrow$ A32 hierarchy.

As illustrated in Fig. 6 (evaluated at a code distance of $d = 7$), we benchmark the primary model against three distinct, aggressively compressed variants: PP (**A1**-A4-A32), PP (A2-**A2**-A32), and PP (A2-A4-**A8**). The results demonstrate a clear performance boundary: further reducing the activation precision in any specific stage causes the network to degrade noticeably, falling below the classical MWPM baseline. Crucially, this evaluation yields a profound physical insight regarding extreme neural quantization. It confirms that forcing the initial spatiotemporal perception stage into a purely binary domain—as seen in the failure of PP (A1-A4-A32)—creates a severe representational bottleneck. Even though the weights are successfully binarized (W1), the dynamic intermediate feature maps strictly require at least 2-bit activations (A2) at the base of the network. This minimal activation precision is mandatory to adequately capture, differentiate, and route the fundamental physical correlations of quantum errors before executing spatial pooling.

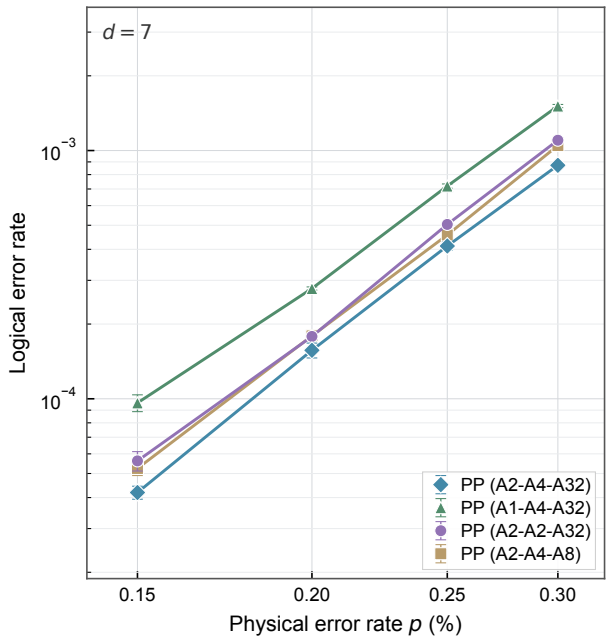


Figure 6: **Ablation study on the precision hierarchy of the** PP **architecture.** The logical error rate is evaluated for a distance $d = 7$ planar surface code across varying physical error rates p. The primary configuration, PP (A2-A4-A32), is benchmarked against three aggressively quantized variants and the classical MWPM baseline. The results demonstrate that extreme binarization of the initial activations (A1) or excessive compression in subsequent pooling stages and final decision (A2 or A8) creates severe information bottlenecks, causing the decoding accuracy to degrade below the classical threshold. The reported configuration in the body part represents the optimal boundary for preserving algorithmic superiority under extreme hardware quantization constraints for surface code.

Precision hierarchy selection and comparison with HGQ [65]. The precision hierarchy is selected through a systematic but empirical procedure rather than an exhaustive search over all possible bit-width assignments. Our goal is to find the lowest hardware precision that preserves the required decoding accuracy. Since both the syndrome input and the logical output are binary, we first explored a fully binary neural network. When its accuracy was insufficient, we fixed all trainable weights at W1 and gradually increased only the activation precision. We increase the precision of a stage only when the validation LER no longer meets the target.

To assess the precision design, we performed two experiments using the same pretrained FP32 backbone as PP on the distance-7 surface code under SI1000 noise with $p = 0.005$. First, with all trainable weights fixed to W1, HGQ searched only the activation precision. The best model reached approximately 70% accuracy, indicating that the extreme W1 setting is difficult to optimize without additional architectural or training constraints.

We then allowed HGQ to search both weight and activation precision under a comparable computational budget. The resulting model uses 906,741,382 EBOPs, approximately 1.02 times the cost of PP, but achieves only 90.92% accuracy, compared with 97.86% for MWPM and 98.06% for PP with the A2-A4-A32 configuration. As shown in Table 8, HGQ uses a highly irregular mixture of weight and activation bit widths, whereas PP uses fully binary weights and a regular stage-wise activation hierarchy.

HGQ searches weight and activation bit widths at a fine-grained level. However, in our matched-cost experiment, the HGQ model does not achieve the accuracy of PP, despite using approximately the same EBOP budget. Nevertheless, HGQ provides a useful alternative quantization strategy and suggests a promising direction for exploring finer-grained and more efficient precision assignments for real-time quantum-error-correction decoding.

Table 8: Weight and activation bit-width distributions for HGQ and PP.

Bit width	HGQ Weight (90.92%)	HGQ Activation (90.92%)	PP Weight (98.06%)	PP Activation (98.06%)
1	9.68%	53.73%	100.00%	0
2	21.83%	6.39%	0	90.93%
3	18.58%	3.72%	0	0
4	7.90%	0.92%	0	6.51%
5	4.40%	0.05%	0	0
6	2.25%	0	0	0
7	0.71%	0	0	0
8	0.38%	0	0	0
9	0.00%	0	0	0
32	0	0	0	2.57%

Fake-End. To explicitly validate the efficacy of the proposed temporal distillation mechanism, we analyze the training dynamics of the PP architecture with and without intermediate supervision. Fig. 7 plots the logical error rate convergence against training steps for the primary PP (A2-A4-A32) configuration. The empirical trajectory demonstrates a substantial advantage: integrating the intermediate distillation (w/ Fake-End) not only visibly accelerates the early-stage convergence rate but also allows the model to settle at a significantly lower final logical error rate compared to the baseline trained exclusively on the final parity label (w/o Fake-End).

This performance divergence stems from two fundamental optimization bottlenecks. First, relying solely on the final parity hard label exacerbates the credit assignment problem; because intermediate physical errors can trivially cancel each other out over multiple syndrome rounds, the final label alone severely dilutes the historical error information. Fake-End supervision resolves this by providing dense, step-by-step guidance, explicitly teaching the network how errors evolve dynamically. Second, and crucially for our hardware-efficient design, these intermediate targets inject rich, localized gradient signals directly into the earlier temporal layers of the network. This continuous gradient flow is instrumental in mitigating the severe optimization instability and gradient mismatch intrinsically caused by the extreme low-bit binarization and the STE, ultimately steering the quantized network out of suboptimal local minima toward a superior algorithmic ceiling.

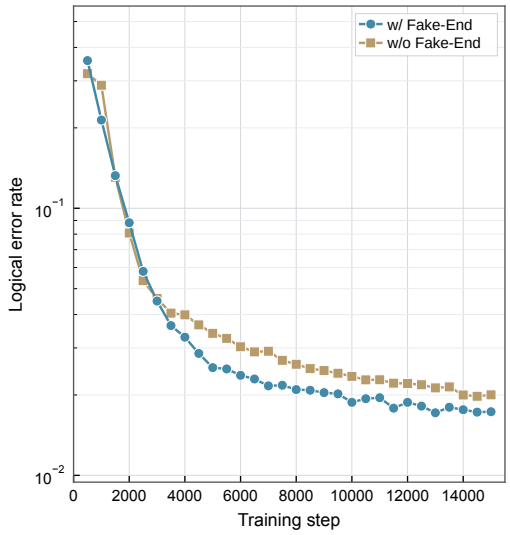


Figure 7: **Convergence analysis of Fake-End supervision.** The training trajectory (logical error rate versus training steps) is plotted for the PP (A2-A4-A32) architecture trained with (w/ Fake-End) and without (w/o Fake-End) intermediate temporal distillation. The results confirm that incorporating intermediate parity guidance systematically mitigates the severe credit assignment and gradient mismatch challenges inherent to deep, extremely quantized networks. This mechanism ensures faster initial convergence and unlocks a strictly superior final decoding accuracy compared to relying solely on the terminal logical outcome.

E.2 Detailed baseline neural decoder results

To further contextualize the algorithmic superiority of our architecture, we systematically evaluate two prominent hardware-oriented neural baselines: NEO-QEC and LATTE. As illustrated in Fig. 8, under rigorous uniform circuit-level depolarizing noise ($p = 0.2\%$), both neural baselines exhibit an anomalous upward trajectory, failing fundamentally to reach the pseudo threshold (i.e., the logical error rate increases as the code distance scales). This catastrophic collapse exposes critical vulnerabilities in prior architectural designs. NEO-QEC, which claims a binarized network structure, originally demonstrated successful error suppression only when evaluated under an oversimplified phenomenological noise model. When subjected to the complex, highly correlated spatiotemporal error syndromes generated by realistic circuit-level noise, its severely limited representational capacity collapses entirely. Similarly, LATTE was originally proposed as a hybrid decoding framework that relies heavily on a classical algorithmic fallback to compensate for the inaccuracies of its highly compressed INT8 neural head. When evaluating this neural decoding head as a standalone solver—stripping away its classical safety net—it proves entirely ineffective at resolving dense error mechanisms. Consequently, these results provide a stark conclusion: prior attempts at deploying heavily quantized or FPGA-targeted neural decoders fundamentally fail to function as viable, standalone solvers under realistic quantum noise environments.

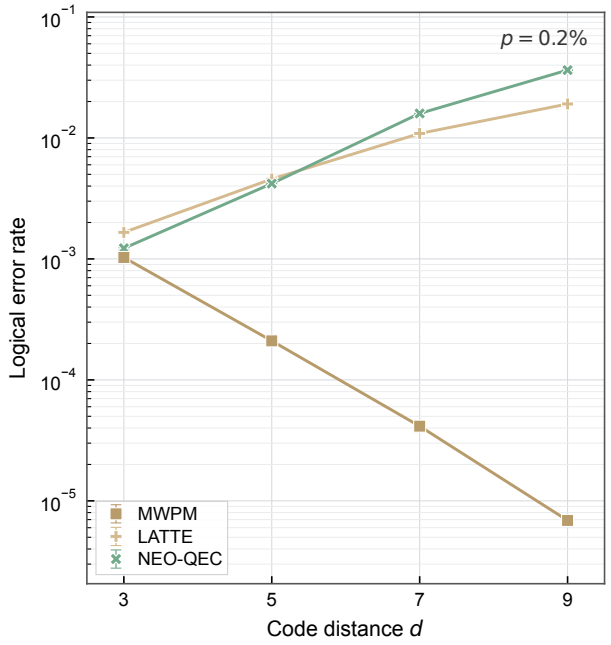


Figure 8: **Performance collapse of prior hardware-oriented neural baselines.** Logical error rate scaling is evaluated for planar surface codes ($d \in \{3, 5, 7, 9\}$) under uniform circuit-level noise at $p = 0.2\%$. While the classical MWPM baseline effectively suppresses errors as the code distance scales, the standalone neural baselines, LATTE and NEO-QEC, categorically fail to reach the pseudothreshold. Their anomalous upward trajectories demonstrate that prior heavily quantized architectures lack the intrinsic representational capacity to resolve complex circuit-level errors without relying on classical algorithmic fallbacks or oversimplified noise assumptions.

E.3 Additional BB-code instances

We additionally evaluate PP on the $[\![30, 2, 5]\!]$ and $[\![48, 4, 8]\!]$ BB codes, together with the original $[\![72, 12, 6]\!]$ code. These instances cover different block sizes, numbers of logical qubits, and code distances. All results are evaluated at $p = 0.5\%$ under the same circuit-level noise and evaluation protocol. Table 9 reports the logical error rate per round per logical qubit, in percent.

Table 9: Logical error rate per round per logical qubit (in percent) for three BB-code instances under circuit-level uniform depolarizing noise at $p = 0.5\%$.

Decoder	$[\![30, 2, 5]\!]$	$[\![48, 4, 8]\!]$	$[\![72, 12, 6]\!]$
PP (A2-A4-A32)	0.231	1.103	0.888
PP (A4-A4-A32)	0.156	0.621	0.406
PP (FP32)	0.143	0.603	0.194
BP-OSD	0.204	1.761	0.519
Relay-BP	0.163	0.669	0.440

Across all three BB instances, the full-precision PP obtains lower LER point estimates than BP-OSD and Relay-BP. More importantly, W1 PP (A4-A4-A32) remains competitive after extreme quantization and also gives lower LER point estimates than both baselines on all three tested codes. Reducing the early perception stage to A2 causes a clear accuracy loss on the $[\![30, 2, 5]\!]$ and $[\![48, 4, 8]\!]$ codes, consistent with the result for $[\![72, 12, 6]\!]$. Thus, the need for A4 perception is not specific to the original $[\![72, 12, 6]\!]$ instance, although we do not claim that A4-A4-A32 is universally necessary for every BB or qLDPC code.

E.4 Training dynamics

As highlighted in our architectural design, optimizing a deeply quantized neural decoder from scratch is notoriously difficult due to catastrophic gradient mismatch. To empirically validate the necessity of our progressive precision-annealed curriculum, Fig. 9 visualizes the training dynamics of the target PP (A2-A4-A32) model under different initialization strategies.

The proposed curriculum systematically decomposes the optimization trajectory into five sequentially constrained phases: (1) An unconstrained FP32 initialization; (2) Weight binarization in the early perception stage (W1A32-W32A32-W32A32); (3) Initial activation compression (W1A8-W32A32-W32A32); (4) Intermediate structural quantization (W1A4-W1A4-W1A32), which serves as the terminal precision target for the complex BB codes; and finally (5) The extreme hardware-oriented configuration (W1A2-W1A4-W1A32) targeted for surface codes.

The empirical trajectory in Fig. 9 reveals several profound optimization insights. First, attempting to train the extreme low-bit configuration directly from a random initialization (From scratch) fails catastrophically; the severe loss of gradient fidelity traps the network in poor local minima, forcing an early termination well above the classical MWPM baseline. In stark contrast, the progressive curriculum (From progressive) successfully guides the network downward. At each transition boundary (marked by dashed lines), loading the converged parameters into a tighter precision subspace naturally induces a sudden accuracy drop (visible as sharp spikes). However, because the structural feature routing is already established, the network reliably recovers and surpasses its previous performance ceiling. Furthermore, the exponentially widening intervals required to reach convergence at later stages (e.g., from Stage 4 to Stage 5) physically manifest the extreme difficulty of optimizing ultra-low-bit spaces, ultimately proving that this sequential warm-start mechanism is an absolute prerequisite for deploying extreme quantization.

E.5 Surface code FPGA HLS synthesis

To validate the hardware feasibility of the deployed surface-code PP (A2-A4-A32) decoder, we synthesize the $d = 9$, $C_{\text{feat}} = 128$ design using Vitis HLS 2025.2, targeting the AMD Versal Premium VP1902 FPGA with a 400 MHz target clock. The synthesized design follows the module partition in Table 2. Binary-weight convolutional and linear layers are implemented without general multipliers; the A2 backbone and A4 readout stages are mapped primarily to LUT logic, while the final A32 decision layer is kept as the high-precision arithmetic stage. Lookup-based embeddings and causal temporal states are stored in on-chip memory.

A key implementation choice is that W1A2 convolutions are not lowered to generic low-bitwidth MAC arrays. Since the weights are binary, each activation term contributes either the positive or negative value of a 2-bit activation code. We therefore implement each output channel as a full-fan-in reduction over all spatial or temporal taps and input channels: the binary weights select the signs

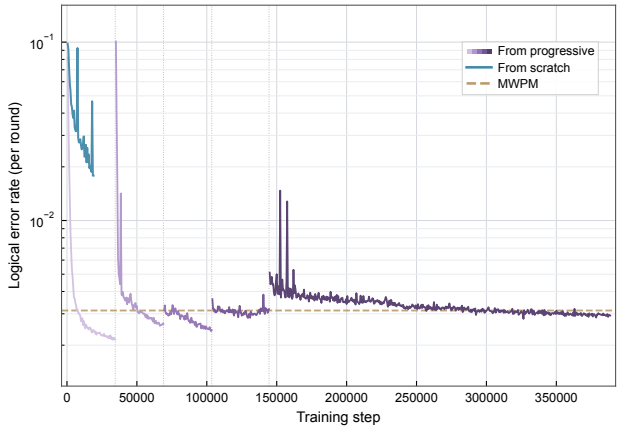


Figure 9: **Training dynamics of the progressive precision-annealed curriculum.** The logical error rate is plotted against training steps, comparing direct quantization (From scratch) versus our proposed five-stage curriculum (From progressive). Vertical dashed lines denote the boundaries of precision reduction steps (e.g., transitioning from FP32 down to the target W1A2-W1A4-W1A32 configuration). While training from scratch suffers from immediate gradient collapse, the progressive approach ensures that the accuracy degradation at each quantization step (visible as intermittent spikes) is strictly bounded and fully recoverable. The increasingly prolonged convergence phases at lower precisions highlight the severe optimization challenges successfully bypassed by this curriculum.

Table 10: Per-module HLS synthesis results for the surface-code PP (A2-A4-A32) decoder at $d = 9$ on VP1902. All compute modules achieve $\mathrm{II} = 1$.

Module	Precision	LUT	FF	DSP	BRAM18	Latency (CC)	II
	W1 lookup						
Local embedding + stem	W1A2	$\sim$ 220,370	$\sim$ 47,038	0	64	$\sim$ 35	1
Grid scatter	–	$\sim$ 6,144	$\sim$ 3,072	0	0	$\sim$ 5	1
TCN_1	W1A2	$\sim$ 139,748	$\sim$ 51,674	0	0	$\sim$ 28	1
SRes_1	W1A2	$\sim$ 1,767,619	$\sim$ 510,605	0	0	$\sim$ 193	1
SRes_2	W1A2	$\sim$ 1,767,619	$\sim$ 510,605	0	0	$\sim$ 193	1
SRes_4	W1A2	$\sim$ 1,767,619	$\sim$ 510,605	0	0	$\sim$ 193	1
TCN_2	W1A2	$\sim$ 139,748	$\sim$ 51,674	0	0	$\sim$ 28	1
Data projection	W1A4	$\sim$ 558,174	$\sim$ 203,168	0	0	$\sim$ 30	1
Row pooling + row head	W1A4	$\sim$ 24,136	$\sim$ 109,476	0	0	$\sim$ 10	1
Final decision	W1A32	$\sim$ 98,160	$\sim$ 263,826	1,533	0	$\sim$ 6	1
Total	–	$\sim$ 6,489,337	$\sim$ 2,261,743	1,533	64	–	–

of the A2 terms, and the selected terms are accumulated by a balanced adder tree. This avoids per-tap serial accumulation and exposes the full reduction parallelism to HLS, which is critical for the latency-oriented VP1902 implementation.

Table 10 reports the per-module HLS synthesis results. All compute modules achieve initiation interval $\mathrm{II} = 1$. The local embedding, grid scatter, temporal blocks, spatial residual blocks, data projection, and readout modules use the same names as in Table 2; this keeps the synthesis report aligned with the deployed decoder rather than with an implementation-specific HLS file hierarchy.

Table 11 summarizes the total resource utilization against the VP1902 device capacity. The expected resource profile follows the precision pyramid: the A2 and A4 stages primarily consume LUTs, while DSP usage is concentrated in the final A32 decision layer. This separation is important for keeping the dominant spatiotemporal feature extraction workload inside the abundant logic fabric.

In continuous QEC operation, the decoder runs as an incremental round-streaming pipeline. At each measurement cycle, it consumes one new syndrome round rather than recomputing the full T_{det} history. For $d = 9$, this round corresponds to the active surface-code detector positions used by the deployed decoder. The causal TCN states are stored in on-chip circular buffers and reused across cycles. Table 12 combines the relevant spatial, temporal, and readout stages to report the

Table 11: Total resource utilization of the surface-code PP (A2-A4-A32) decoder on VP1902.

Resource	Usage	VP1902 total	Utilization
LUT	$\sim$ 6,489,337	8,460,288	$\sim 76.70\%$
FF	$\sim$ 2,261,743	16,920,576	$\sim 13.37\%$
DSP58	1,533	6,864	$\sim 22.33\%$
BRAM	64	13,616	$\sim 0.47\%$
URAM	0	2,200	0.0%

Table 12: Incremental inference latency for the surface-code PP (A2-A4-A32) decoder at $d = 9$ on VP1902 at 400 MHz. The decoder processes one new syndrome round per QEC cycle using buffered causal states.

Module	Clock cycles	Latency (μs)
Local embedding + grid scatter	~ 40	~ 0.100
TCN_1 incremental update	~ 28	~ 0.070
Spatial residual encoder (SRes_1–SRes_4, pipelined)	~ 193	~ 0.483
TCN_2 incremental update	~ 28	~ 0.070
Data projection + row readout + final decision	~ 46	~ 0.115
Total incremental latency	~ 335	~ 0.8375

total incremental latency for processing one new syndrome round, with clock cycles converted to microseconds at 400 MHz. In the pipelined schedule of Table 12, the spatial residual encoder entry represents the steady-state incremental latency rather than the sum of the standalone latencies reported separately for SRes_1, SRes_2, and SRes_4 in Table 10, because these blocks process different syndrome rounds concurrently after pipeline filling.

E.6 Vivado place-and-route results

We perform a complete Vivado place-and-route analysis for both the $d = 9$ surface-code decoder and the $[[72, 12, 6]]$ BB-code decoder. The resulting implementation-level resource usage, timing status, and decoding latency are reported below for both code families.

Implementation setting. All results below are obtained with Vivado 2025.2 targeting the AMD Versal Premium VP1902 (xcvp1902-vsva6865-2MP-e-S) under a 400 MHz clock constraint, running the complete implementation flow: synthesis, logic optimization, placement, physical optimization, routing, and post-route physical optimization. The complete campaign consumed over 400 CPU-hours of Vivado runtime.

Surface code ($d = 9$). The integrated design currently achieves WNS -0.031 ns against the 2.5 ns constraint, corresponding to approximately 395 MHz. We note that timing closure at this scale is itself an iterative optimization process: with additional physical-optimization passes and more compute time, the remaining slack can be further reduced.

BB code ($[[72, 12, 6]]$). The BB decoder is fully placed and routed and closes timing at 400 MHz (post-route WNS $+0.099$ ns).

Latency. The measured decoding latency is **0.89 μs** for the $d = 9$ surface-code decoder and **141.2 μs** for the BB-code decoder, within the 1 μs superconducting budget and the 1–10 ms trapped-ion/neutral-atom budget, respectively.

Table 13: Post-route Vivado resource utilization for the $d = 9$ surface-code decoder on VP1902.

Block	LUT	FF	DSP58
Embedding + stem	630 ($< 0.01\%$)	–	0
TCN stack ($2\times$)	1,887,616 (22.3%)	823,344 (4.9%)	0
SRes stack ($3\times$)	3,548,016 (41.9%)	2,597,688 (15.4%)	0
Readout	174,596 (2.06%)	107,011 (0.63%)	0
Final decision	1,186 (0.01%)	99,401 (0.59%)	1,022 (14.9%)
Top-level interconnect	71,813 (0.85%)	269,194 (1.59%)	0
Total	**5,683,857 (67.2%)**	**3,896,638 (23.0%)**	**1,022 (14.9%)**

Table 14: Post-route Vivado resource utilization for the $[[72, 12, 6]]$ BB-code decoder on VP1902.

Block	LUT	FF	DSP58
Check-token embedding	–	–	0
TCN stack ($2\times$)	594,442 (7.0%)	564,842 (3.3%)	0
PolyGraphRes stack ($3\times$)	2,587,854 (30.6%)	1,395,862 (8.3%)	0
Check-to-data + logical readout	676,878 (8.0%)	420,246 (2.5%)	0
Final decision	231,102 (2.7%)	278,018 (1.6%)	1,258 (18.3%)
Top-level interconnect	325,008 (3.8%)	746,216 (4.4%)	0
Total	**4,415,284 (52.2%)**	**3,405,184 (20.1%)**	**1,258 (18.3%)**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper's contributions and scope are correctly described in both the abstract and the introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations of our work are discussed in the final section of the main paper.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer:[N/A]

Justification: The paper does not include theoretical results or proofs; its contributions are algorithmic and empirical in nature.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The appendix provides all necessary architectural, training, and evaluation details to reproduce the main claims.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code and data is available.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all training and test details in Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The main figures include error bars.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experimental results section offers a comprehensive description of the required computational resources.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: The paper focuses on a foundational technology for fault-tolerant quantum computing and does not explicitly discuss its broader societal impacts, either positive or negative.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases a model for quantum error correction decoding. This technology poses no high risk for misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the original works for all datasets and code bases it uses . All assets are open-source for research purposes, and their terms of use have been respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper will provide data and code. The paper itself provides extensive documentation of the model, training procedures, and experiments in the appendices, which can be refined into complete documentation.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core methodology of the paper is a novel neural network architecture for quantum error correction and does not involve the use of LLMs.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.