In-pixel AI for lossy data compression at source for X-ray detectors

FERMILAB-PUB-23-598-ETD-PPD

DOI: 10.1016/j.nima.2023.168665

Accepted Manuscript

In-pixel AI for lossy data compression at source for X-ray detectors

Manuel B. Valentin[a], Giuseppe Di Guglielmo[a,b,*], Danny Noonan[b], Priyanka Dilip[b], Panpan Huang[a], Adam Quinn[b], Thomas Zimmerman[b], Davide Braga[b], Seda Ogrenci[a], Chris Jacobsen[a], Nhan Tran[a,b], Farah Fahim[a,b]

[a]*Northwestern University, 2145 Sheridan Rd, 60208, Evanston, IL, USA*
[b]*Fermi National Accelerator Laboratory, Pine & Kirk St, 60510, Batavia, IL, USA*

Abstract

Integrating neural networks for data compression directly in the Read-Out Integrated Circuits (ROICs), i.e. the pixelated front-end, would result in a significant reduction in off-chip data transfer, overcoming the I/O bottleneck. Our ROIC test chip (AI-In-Pixel-65) is designed in a 65nm Low Power CMOS process for the readout of pixelated X-ray detectors. Each pixel consists of an analog front-end for signal processing and a 10b analog-to-digital converter operating at 100KSPS. We compare two non-reconfigurable techniques, Principal Component Analysis (PCA) and an AutoEncoder (AE) as lossy data compression engines implemented within the pixelated area. The PCA algorithm achieves 50× compression, adds one clock cycle latency, and results in a 21% increase in the pixel area. The AE achieves 70× compression, adds 30 clock cycle latency, and results in a similar area increase.

1. Introduction

Ptychography is an imaging method that delivers a spatial resolution limited not by any optics used, but by the scattering strength of the object under study. This is especially important in X-ray microscopy, where the numerical aperture of the best optics is quite low compared to that obtained with visible light.

*Corresponding author
Email address: gdg@fnal.gov (Giuseppe Di Guglielmo)

Ptychography works by collecting far-field diffraction patterns as a limited-area coherent beam is moved to different locations on an extended object, with object reconstruction obtained using iterative phase retrieval or nonlinear optimization methods. Obtaining faster frame-rate detectors is essential for improving the throughput of X-ray ptychography [1], especially as diffraction-limited storage rings are providing hundredfold gains in coherent flux [2].

Typical pixelated readout ICs for X-ray detectors have 200×200 pixels per chip, utilize 10-12b ADCs for digitizing data, and need to operate continuously without deadtime at high frame rates up to 1 Mfps, thus generating approximately $\geq$ 0.5 Tbps of data. The main bottleneck at this stage is the off-chip data transfer. If this data could be reduced by a factor between $50\times$ and $100\times$, then 1Mfps readout can be achieved with 2-4 multi-gigabit links.

Alternative techniques, such as data sparsification using zero suppression, require significant overhead associated with pixel address (approximately 18-20b per 12b data for a full reticle chip). Moreover, although simulated data could contain $\sim 97\%$ zeros, the noisy experimental data is closer to $\sim 60\%$ zeros, indicating a much higher occupancy. A breakeven analysis of full-frame imaging vs. zero-suppressed readout for large pixel Read-Out Integrated Circuits (ROICs) shows that zero suppression is no longer useful for occupancies $\geq 40\%$ [3].

Although front-end ROICs have adopted early digitization, such as in-pixel ADCs [4] and TDCs, they still rely on data transfer from pixel to the periphery, on-edge data serialization, and high-speed off-chip data transfer [5]. In pixel detector chips, considerations of geometry and power both contribute to a bottleneck in extracting data from the chip. A pixel detector with millions of pixels generating 10-bit data words at a rate of tens to hundreds of kilohertz can produce 10^{12} bits per second. In particular, the state of the art in electronic links for high energy physics is represented by the lpGBT ASIC developed by CERN, which can read out 10 Gb/s with power dissipation of over 0.5 $Watts$ [6, 7]. Such a pixel detector would require hundreds of lpGBT channels operating in parallel to read it out, which is clearly infeasible from a system perspective. The development of novel integrated photonic links can ease this bottleneck by

providing data transmission at $< 0.5\ pJ/b$ (compared to $10pJ/b$ for state-of-the-art electronic solution) and by using wavelength-division multiplexing (WDM) to replaced dozens of electrical channels with a single optical fiber, increasing feasible readout bandwidth by two orders of magnitude or more [8]. However, significant challenges still remain in the integration and co-packaging of silicon photonics with CMOS readout. On-ASIC data compression provides an alternative solution for addressing the readout bottleneck. In most scientific applications, the rate of features of interest in a data stream is a tiny fraction of the raw data rate, and discarding data which is not "interesting" before it is moved off-chip can dramatically reduce both power and bandwidth consumption. The advancement of both CMOS technology nodes and machine learning algorithms has made on-chip discrimination realistic with only a modest power and area overhead, making data compression useful to both bridge the gap until more efficient link strategies are mature, as well as complement them when they arrive.

In this work, we aim to demonstrate that lossy data compression techniques such as Principal Component Analysis (PCA) and AI/ML-based AutoEncoders (AE) could enable 50× to 80× on-chip data compression as a pathway to overcoming the I/O bottleneck while maintaining the accuracy required for image reconstruction and further scientific analysis. Traditionally, edge AI has constituted the implementation of neural networks (NNs) on FPGAs or digital data concentrator ASICs, which collect and process data from several front-end ROICs. However, integrating NNs directly in the pixelated front-end ROICs would result in a significant reduction in off-chip data transfer.

Our AI-In-Pixel-65 test chip architecture for X-ray detectors includes two 32×32 arrays of pixels with independent readouts after data compression, as shown in Fig. 1. Figure 2 shows the layout of the readout chip with the PCA and AE algorithms integrated into the pixelated areas. The following sections describe the pixel front-end architecture, the algorithm development, some implementation issues, and our co-design solutions.

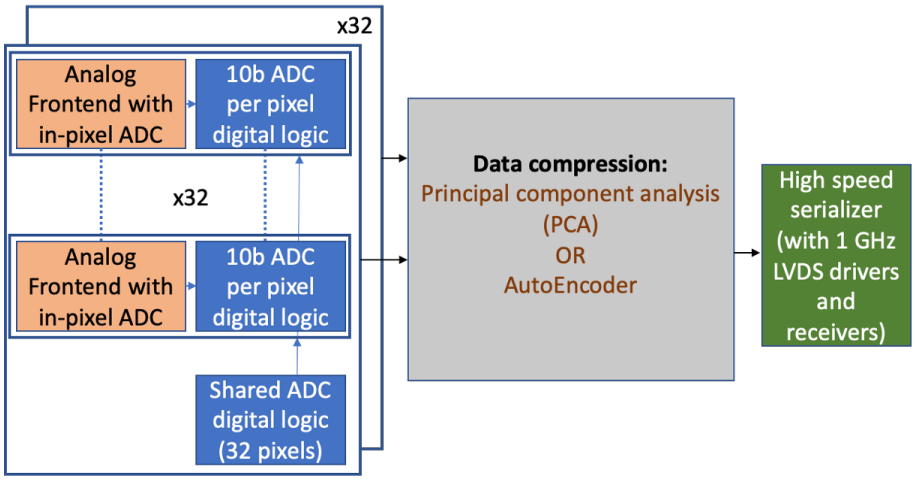

Figure 1: Functional block diagram of AI-In-Pixel-65 test chip with either PCA or AE performing data compression for 1024 pixels

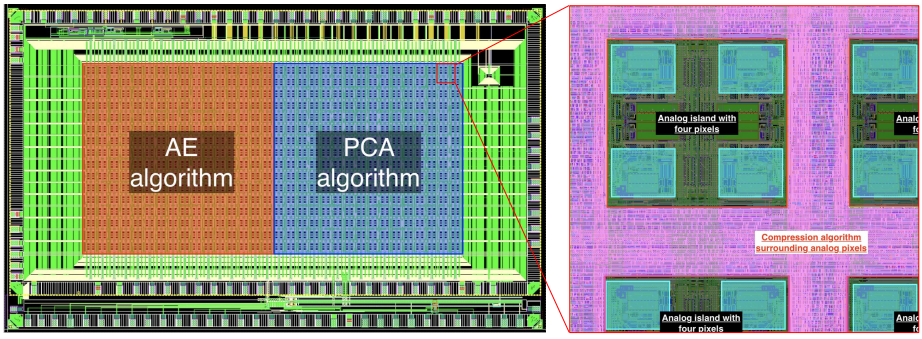

Figure 2: Our ROIC test chip (AI-In-Pixel-65) with the PCA and AE algorithms integrated in the pixelated area. On the right, an highlight of the compression algorithm in digital logic surrounding the analog pixels.

2. Front-end architecture

The AI-In-Pixel-65 analog front-end consists of three stages: a charge-sensitive integrator, correlated double-sampling circuit, and compact 100 KSPS serial SAR ADC [9].

A charge-sensitive integrator is directly connected to the photodiode, converting pulses of charge to voltage. Its feedback capacitor is a 3 fF plate capacitor with the top-metal bump bond forming one plate. The integrator's output is sampled by a correlated double-sampling (CDS) circuit, which suppresses

low-frequency noise and reset noise [10].

The sampled voltage is digitized by a ten-bit serial successive approximation register (SAR) ADC. The ADC generates approximation voltages using a charge redistribution DAC based on two capacitors C_R and C_L. Six binary weighted trim capacitors are used to tune $C_L = C_R$. When not selected, these trim capacitors are bootstrapped by the comparator's input buffer.

The DAC generates each successive bit of the approximation voltage by charging C_R either to V_{ref} or to 0V, then shorting the positive terminals of C_R and C_L together [11]. The final voltage developed at the positive terminal of C_L (the negative input to the comparator) after N phases is given by:

$$\sum_{n=1}^{N} \frac{V_{ref} h_n}{2^{N-n+1}}$$

Where h_n is one if C_R is charged to V_{ref} at stage n and zero if it is discharged. The ADC compares each approximation to the sampled voltage at its positive input to decide the next h_n. C_L is then discharged. Thus, one 10-bit ADC acquisition requires 55 clock cyles: $\sum_{n=1}^{10} n = 45$ cycles to compute 10 approximations, plus 10 discharge/clear cycles. However, the data from one ADC acquisition can be read out while the next sample is being acquired, so no dead time is introduced if total readout time is less than one acquisition period [5].

3. Lossy data compression

Very fast and simple schemes have been proposed [12] for about 2x on-detector-chip data compression with little or no impact on reconstructed image quality [13], and these schemes are similar to those used for compressed storage of ptychographic data in GPUs during image reconstruction [14]. At the same time, more complicated compression methods such as principal component analysis (PCA) have been shown to provide significantly higher compression values for storing ptychographic data for subsequent analysis. Alternatively, it has been shown that one can train neural networks from the association of real-space object features and their far-field diffraction patterns, and subsequently

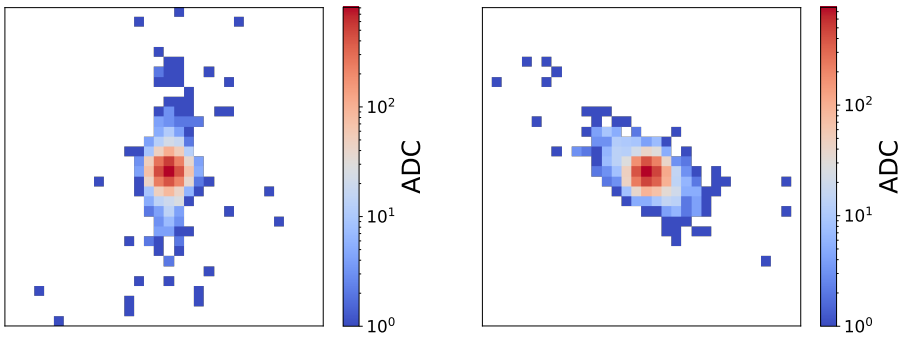

Figure 3: Simulated example diffraction patterns measured by a 10-bit ADC in a 32×32 pixel array.

use these networks to rapidly obtain approximate object reconstructions from diffraction data [15, 16]. Therefore it is worthwhile to consider different options for more-sophisticated on-chip data compression of ptychographic data, both to save on subsequent data storage space but also to reduce the complications and compromises that the circuitry for high-speed per-pixel data transmission involve [17, 18].

Two schemes for lossy data compression are implemented on separate halves of the chip, each compressing the digitized charge values from their own 32×32 pixel arrays. One half of the chip compresses data through a PCA algorithm, while the other compresses via an AI/ML-based AutoEncoder (AE). Figure 3 shows two example patterns from simulations of X-ray scattering as measured by such a pixel array.

3.1. Principal Component Analysis

The aim of PCA is to represent the diffraction pattern images as a linear combination of the features that make up the pattern. The diffraction pattern measured by the array of j pixels, $\mathbf{D}_j$, can be described by the product of a matrix of eigenimages, $\mathbf{R}_{n \times j}$, representing a basis of the diffraction pattern, and the corresponding vector of eigenvalues, $\mathbf{P}_n$:

$$\mathbf{D}_j = \mathbf{P}_n \times \mathbf{R}_{n \times j}$$

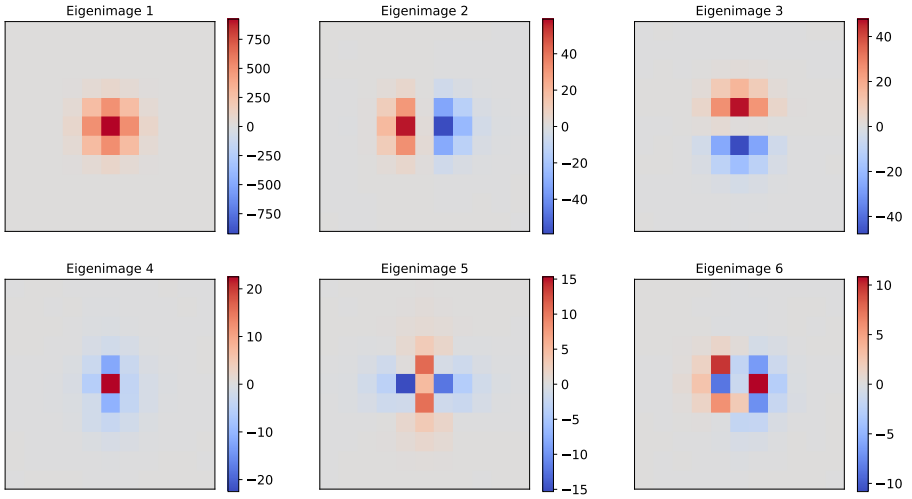

Figure 4: First 6 eigenimages forming the basis of diffraction patterns derived from simulation of a 32×32 pixel array. Images show a zoomed in view of the central 10×10 pixels of the arrays.

Given the matrix of eigenimages $\mathbf{R}_{n\times j}$, the pixel outputs can be described by the n eigenvalues. The eigenimages can be thought of as encoding different features within the diffraction pattern, with the leading eigenimages representing the most prominent features, and progressing to finer and finer details. An example of the first 6 eigenimages for simulated diffraction patterns is shown in Fig. 4. In this basis, the first eigenimage characterizes the amplitude of the central peak in the diffraction pattern, followed by eigenimages which characterize the rotation, spread, and further details of the diffration patterns.

The diffraction patterns are reconstructed through a linear combination of the eigenimages, each weighted by its corresponding eigenvalue. The level of detail and resolution obtained depends on the number of eigenimages/eigenvalues used. Fig. 5 shows an example diffraction pattern from simulation reconstructed using varying number of eigenimages. While reducing the number of eigenvalues transmitted off-chip will result in a lossy reconstruction of the diffraction pattern, ptychography data normally contains a high amount of redundancy. Diffraction patterns are collected from illuminating overlapping locations on a sample, so some levels of loss in the diffraction image can be acceptable while

still maintaining good quality in the reconstructed image of the sample [13].

Since $n \ll j$ typically, this can result in a significant reduction in the data volume if $\mathbf{P}_n$ is calculated on-chip ($\mathbf{P}_n = \mathbf{D}_j \times \mathbf{R}^{-1}_{j\times n}$) and transmitted in place of $\mathbf{D}_j$. The weights used for the inverse of the eigenimage matrix ($\mathbf{R}^{-1}_{j\times n}$) were determined from simulated diffraction patterns. The nature of the x-ray diffraction patterns in ptychography is such that the types of patterns obtained through imaging of different objects are quite similar, and the diffraction patterns obtained from one object can be represented using the basis of eigenimages derived from another object. This means that the elements of the $\mathbf{R}^{-1}_{j\times n}$ matrix can be re-used, and thus hard-coded in the chip.

The number of eigenvalues used and the precision of the weights of the $\mathbf{R}^{-1}$ matrix and eigenvalues were optimized for the performance of the reconstruction of the patterns and the amount of resulting data compression. With more eigenvalues transmitted off-detector (and thus, a larger the $\mathbf{R}^{-1}$ matrix), finer granularity details of the diffraction patterns are encoded. The trade-off between the amount of compression and the the loss of detail in the diffraction patterns is evaluated by compressing and decompressing the diffraction patterns from simulated X-ray ptychography measurements while varying the number of eigenvalues. These diffraction patterns are then used to perform a ptychographic image reconstruction of the initial object from the simulation. The level of detail in the reconstruction is quantified by computing the Fourier ring correlations (FRC) [19] between the reconstructed image and the initial image used to simulate the diffraction patterns. The FRC quantifies the similarity of two images across varying spatial resolutions. In this application, it was found that the spatial resolution of the reconstruction was impacted when fewer than 30 eigenvalues were used, but reconstructing with more than 30 eigenvalues resulted in only small improvements in resolution, so a value of $n = 30$ was chosen. An example of the FRC curves for performing ptychographic image reconstruction with the data compressed using a PCA algorithm with varying number of eigenvalues is shown in Fig. 6. The precision of the eigenvalues was also varied, and it was found that with 30 eigenvalues at seven-bit precision provide good image

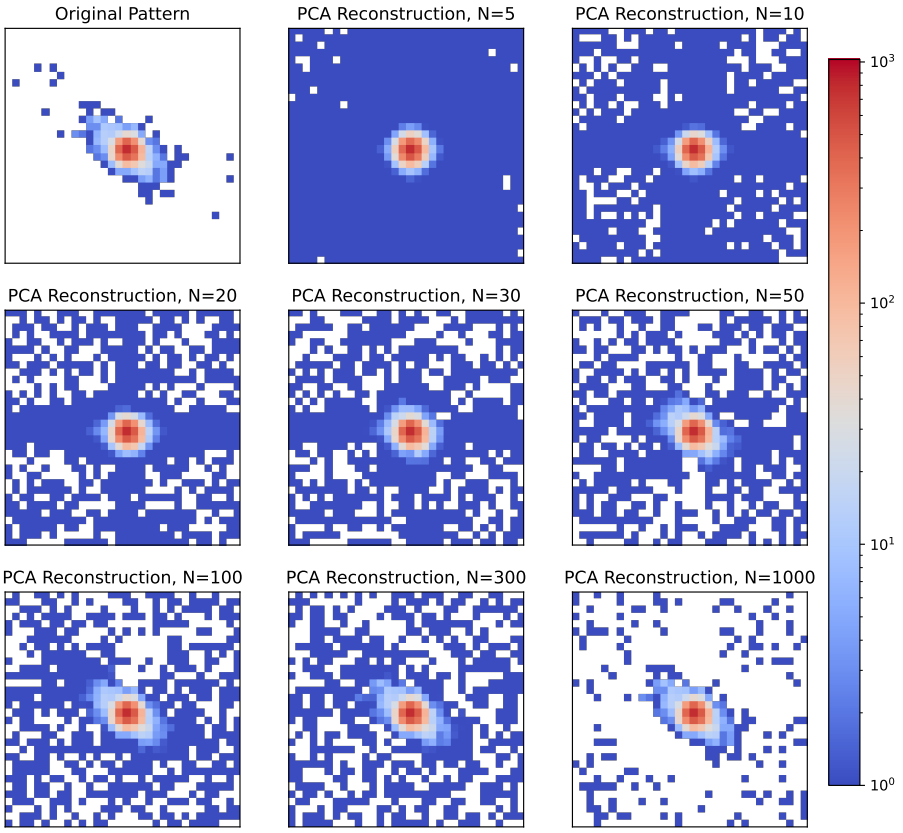

Figure 5: Reconstructed diffraction patterns from PCA algorithm with different number of eigenimages/eigenvalues. The top left panel shows the original diffraction pattern from simulation.

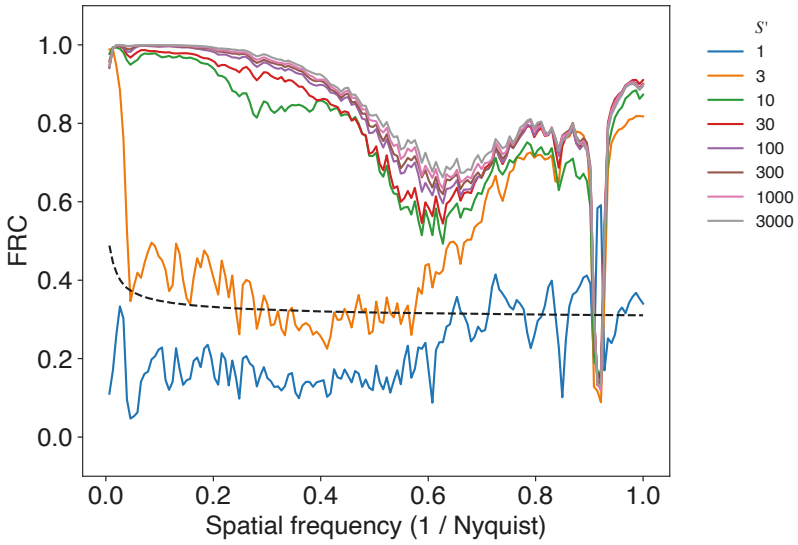

Figure 6: FRC curves for reconstruction of images where diffraction patterns have been compressed with PCA algorithms using different numbers of eigenvalues, denoted by S'. Below 30 eigenvalues, the performance of the reconstruction begins to degrade. The figure has been reproduced from [20].

reconstruction integrity while providing a 50× compression of the data volume.

3.2. AutoEncoder

The AE neural network compresses data into a smaller latent space. The design of the AE network was chosen to closely resemble that of the PCA algorithm, with a single fully connected dense layer being used to encode the data into a 30-value latent space representation which can be transmitted off-chip, with a second dense layer decoding to the latent space back into a representation of the original image. The two dense layers are functionally just matrix multiplications, such that the encoder and decoder layers perform the same functions as the $\mathbf{R}^{-1}$ and $\mathbf{R}$ matrices of the PCA algorithm.

The primary difference between the algorithms is in the way the weights of the matrices are determined. While in PCA, the weights are calculated from the eigenvalues of the diffraction images, the autoencoder uses machine learning to determine the weights. The AE was trained using QKeras [21], which allows

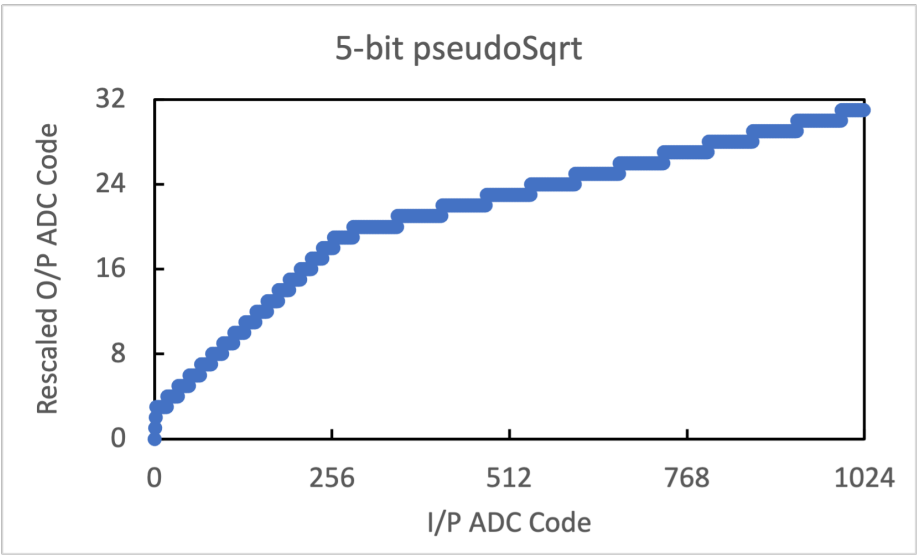

Figure 7: Preprocessing of digitized data before AE network. The 10-bit ADC values are reduced to 5-bits through a remapping that approximates a square root function.

for quantization-aware training of the neural network, optimizing the weights aware of the constraints on the precision of the weights and outputs. Similar to the PCA algorithm, the weights of the AE can be re-used to encode diffraction patterns from different objects, and thus can be hard-coded in the chip. A second difference between the two algorithms comes from a preprocessing of the digitized data before the matrix multiplication in the AE, where the 10-bit ADC values are reduced to five-bit precision through a simplified remapping that approximates a square root function, as shown in Fig. 7.

Similar to the optimization of the PCA algorithm, the precision of the weights and outputs of the AE algorithm were studied. For algorithms trained with different weight and output precision, simulated diffraction patterns were encoded and decoded, and used to perform a ptychographic image reconstruction. The FRC of the different precision options was compared, and it as found that an AE algorithm trained to transmit 30 values at five-bit precision each was able to achieve a 70× compression of the original data, while achieving comparable image reconstruction resolution to the PCA algorithm. An exam-

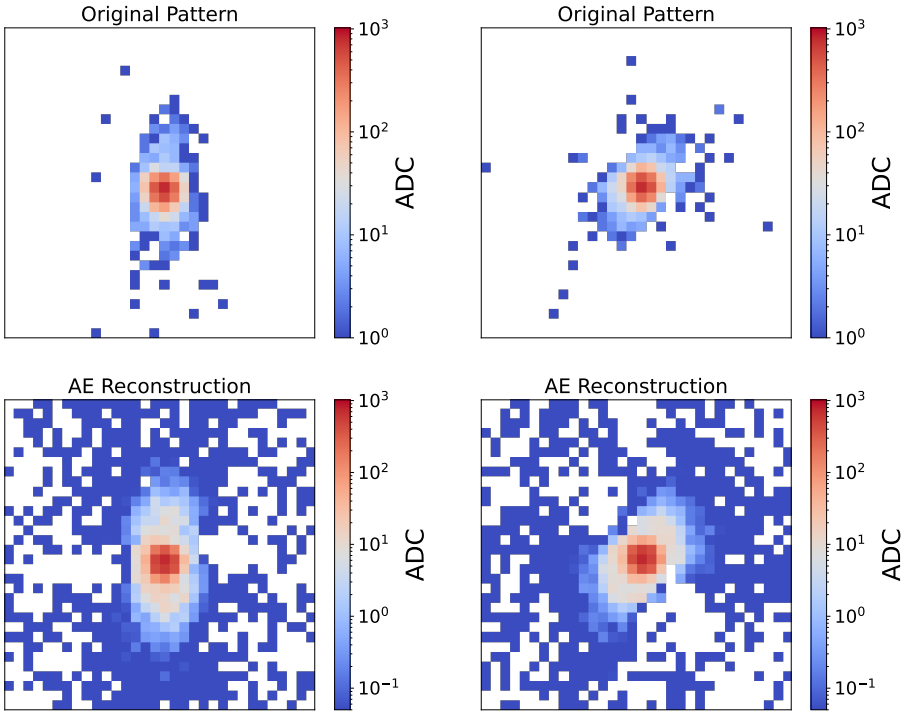

Figure 8: Reconstructed diffraction patterns from autoencoder algorithm. The diffraction patterns on the top row show the original simulated diffraction patterns, and the bottom row show the reconstruction of the same patterns after compressing and decompressing with the autoencoder.

ple of the diffraction pattern outputs obtained before and after compression and decompression with the autoencoder is shown on Fig. 8.

3.3. Comparison

The two algorithms for lossy data compression were designed to have very similar implementations on the chip, with both being simply matrix multiplications to reduce outputs to 30 values from 1024, with the differences between the algorithms being in the precision and specific values that go into the weights. In the PCA algorithm, the leading eigenimages of the $\mathbf{R}$ matrix will have similar sparsity as the diffraction patterns themselves, so the weights used on the chip for the $\mathbf{R}^{-1}$ matrix will tend to have a large number of zeros, which can be removed during the synthesis steps. By contrast, very few of the weights of the

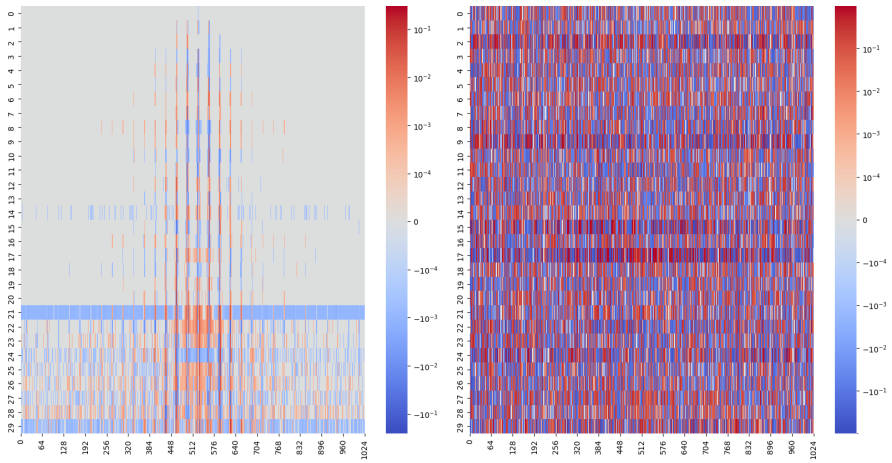

Figure 9: Distribution of the weights for the PCA (left) and AE (right) algorithms. Due to the nature of the training of these algorithms, the PCA tends to have a larger fraction of zero-valued weights compared to the AE.

machine learning-trained AE tend to be zeros. Figure 9 shows the distribution of the weights in the two algorithms. Our analysis of these weights, from the aforementioned approaches of eigenvalue calculation for the PCA model and quantization-aware training for the AE model, showed that the weight sparsity (i.e., quantity of zero-valued weights) was nearly 77.98% for the former and just 8.69% for the latter. Additionally, the scale of the inputs going into the two algorithms differs, with the PCA algorithm receiving the full 10-bit ADC from each pixel and the ADC value used in the AE first being scaled from 10-bits to 5-bit precision. The quantization aware training of the AE algorithm allowed for making better use of bits, and performing the operations on the lower precision inputs, providing saving in the number of computations required.

4. Design methodology

Our design methodology leverages high-level synthesis (HLS) and hls4ml to provide a shorter design cycle and similar quality of result compared to register-transfer level (RTL) flows [22, 23]. hls4ml is an open-source framework for the codesign of optimized neural networks and deployment on custom hardware (ASIC) and field-programmable gate arrays (FPGAs) [24]. At its core, hls4ml

translates machine-learning models from common open-source software frameworks such as Tensorflow into a RTL implementation using HLS tools.

In hls4ml, a designer can trade off the performance (i.e., latency and throughput) and resource utilization for a model by varying the parallelization of the algorithm via several configuration parameters. For example, the *reuse factor* (RF) parameter controls how many times each multiplier resource is used in the final hardware implementation: a designer with the goal of low latency will choose the lowest RF value.

In this work, we used hls4ml. Additionally, we combined quantization-aware training in QKeras [21] with model- and hardware-centric optimizations to find models that simultaneously accomplish the goals of high accuracy, low latency, and low area. hls4ml coupled with the industry standard Siemens Catapult HLS (ver. 2022.1) tool allowed us to explore the cost and performance trade-offs of various architectural hardware implementations for our ML models.

We decided on a reduced-latency and highly-parallel implementation for our accelerators, where 1024 multiplications get performed per clock cycle. We used a fixed-point representation (Siemens `ac_fixed` [25]) for the input, intermediate, and output parameters of our ML model designed with hls4ml. This choice provided us with a high degree of flexibility for exploring the area and accuracy trade-off of the ML-model hardware implementations obtained with Catapult HLS. Fig. 11 shows the interface and bit-precision for both the PCA and AE accelerators, while Sec. 6 describes additional HLS-driven architectural choices.

5. Implementation issues

As stated earlier, the AE algorithm was built using a fully-dense architecture, among other factors, so that the final implementations of this algorithm and the PCA algorithm would resemble each other the most. The high accuracy in image reconstruction that we obtained for the training datasets for the final versions of both these algorithms is a clear indicator of the suitability of the fully-dense architectures chosen for this problem.

However, from a place and route point of view, dense connections pose an extra challenge when analog signal processing and digital data processing is interleaved creating a distributed network of placement and routing blockages. The interconnectivity of inputs and weights in the algorithm specification is expected to be converted into connections and components at the hardware level. Thus, as the number of pixels grows, the number of connections between them grows exponentially, and so does the average distance between two connected pixels. This, in turn, leads to congestion issues during routing as the requirement for more multiply and accumulates (MACs) that perform the matrix multiplication between evermore far-away pixels also grows.

Figure 10 shows an example of the amount of congestion these designs can display after routing (in this case, the AE algorithm). This congestion commonly translates into unwaivable DRC violations at the end of the digital routing stage, and thus require active manual intervention in order to be fixed. The most common and straightforward methodology applied in such cases is to increase the total size of the design, in case area is not a strict constraint. Nevertheless, this solution might not be enough in some cases. Figure 10 shows the congestion our design displayed after routing, even after increasing the pitch between pixels, and thus the area by a factor of 125%. The initial logic occupancy is only 43%, and thus the congestion is entirely due to routing signals across the matrix. Thus, in order to solve the issues with congestion we performed the co-design solution introduced in the Section 6.

6. Co-design solutions for integrated signal and data processing

We used two complementary approaches to tackle the congestion issues. First, we increased the pitch size, as described in Sec. 5. This section instead describes our co-design solutions with Catapult HLS to produce implementations that are easier to place and route in a pixelated area. Fig. 11 highlights the main architectural choices for the two HLS designs. For the AE implementation, we had initially designed the weights of the dense layer as a two-dimensional array where the channel dimension was last, as shown for the buffer at line 7

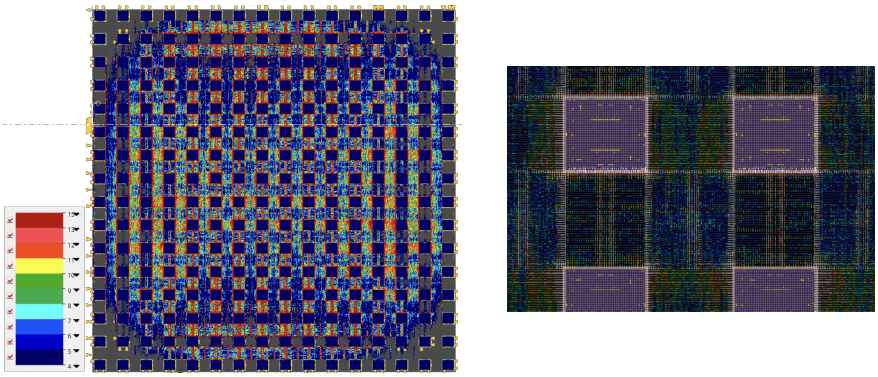

Figure 10: Example of routing congestion issues faced during PnR for the AE algorithm, when using a floorplan with an increase in area by 125% full compared to a baseline implementation without compression logic. We report the entire chip and a closeup view. Here, colors represent different levels of congestion (e.g., the bottom-most entry of the legend, in dark blue, represents a density of wires that is 104% above the limit for any given specific space, thus causing congestion). It is very noticeable how the congestion spreads across the entire chip, while only the pixels at the edges show no issues. This leads to the aforementioned congestion and the inability of the tool to route the design without causing a large amount of unwaivable DRC violations.

and for-loops at lines $17-21$ of Listing 1. A channel-last layout of the data is a default in machine-learning frameworks such as TensorFlow that we used to train the model.

We explored moving the channel dimension first to reduce the amount of control logic as shown at line 9 of Listing 2. Restructuring the weight buffers allowed us to perform 1,024 multiplication in parallel and efficiently pipeline them across each of the 30 channels (lines $13-24$ in Listing 2). We also increased the modularity of the HLS-generated RTL code by partitioning the fully-parallel 1,024-input multiplier accumulator (MAC) into 256 four-input MACs and an additional adder tree in the top-level module (lines $17-22$). For each of those smaller MACs, we wired four neighboring pixels and constrained the place and route tool to position the associated logic in their proximity, as shown in Fig. 12.

We adopted a different solution for the HLS-based implementation of the PCA design after observing the highly-sparse weight matrix. In this case, we inlined all of the HLS-code functions and unrolled all the loops. This approach allows Catapult HLS to perform constant propagation aggressively, remove all of the logic that depends on a multiply-by-zero, and finally simplify most of the

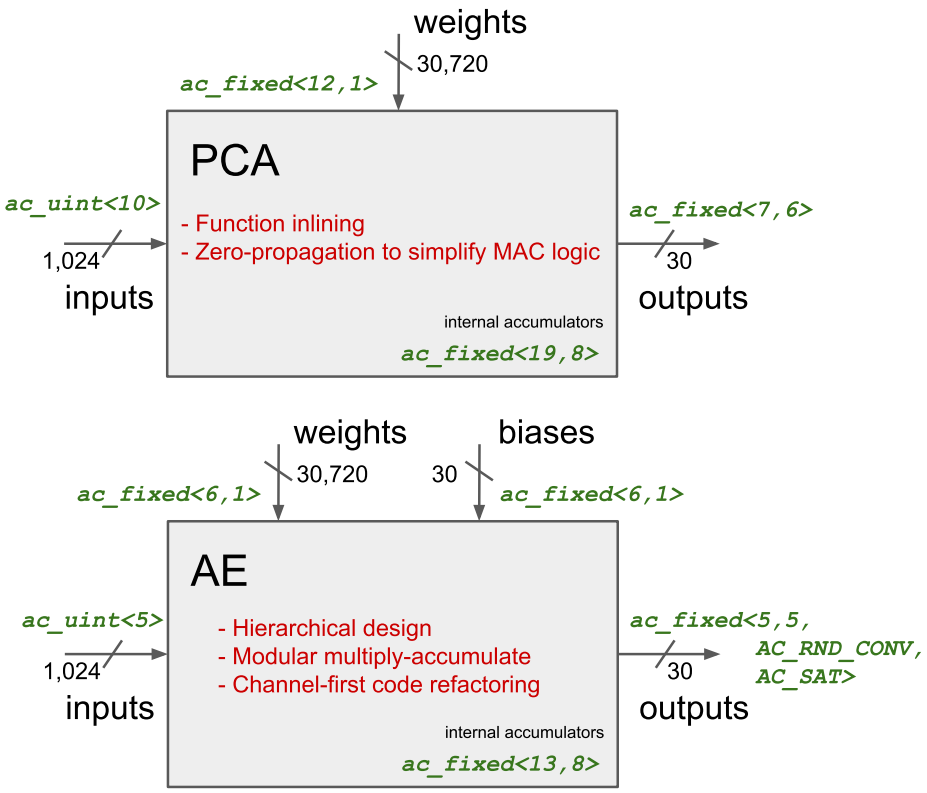

Figure 11: PCA and AE bit-precision and HLS architectural choices.

```
#define H 32 // row count
#define W 32 // column count
#define C 30 // channel count (each channel maps to a multiplier)

void ae_top(
  input_t inputs[H*W],
  weights_t weights[H*W][C], // channel is last
  biases_t biases[C],
  outputs_t outputs[C]) {

  accum_t accum[C];

  for (u32 j = 0; j < C; j++) {
    accum[j] = biases[j];
  }

  for (u32 i = 0; i < H*W; i++) {
    for (u32 j = 0; j < C; j++) {
      accum[j] += inputs[i] * weights[i][j];
    }
  }

  for (u32 j = 0; j < C; j++) {
    outputs[j] = accum[j];
  }
}
```

Listing 1: Initial AE specification for HLS.

```
#define H 32 // rows
#define W 32 // columns
#define C 30 // multipliers

#define B 4 // adders

void ae_top(
  input_t inputs[H*W],
  weights_t weights[C][H*W], // Multiplication is first
  biases_t biases[C],
  outputs_t outputs[C]) {

  for (u32 j = 0; j < C; j++) { // Pipeline
    accum_t accum = biases[j];
    for (u32 i = 0; i < (H*W)/B; i++) { // Unroll
        accum_t sub_accum = 0.0;
        for (k = 0; k < B; k++) { // Submodule, unroll
            sub_accum += inputs[i*B+k] *
                         weights[j][i*B+k];
        }
        accum += sub_accum;
    }
    outputs[j] = acc;
  }
}
```

Listing 2: Refactoring for AE specification to increase modularity in HLS.

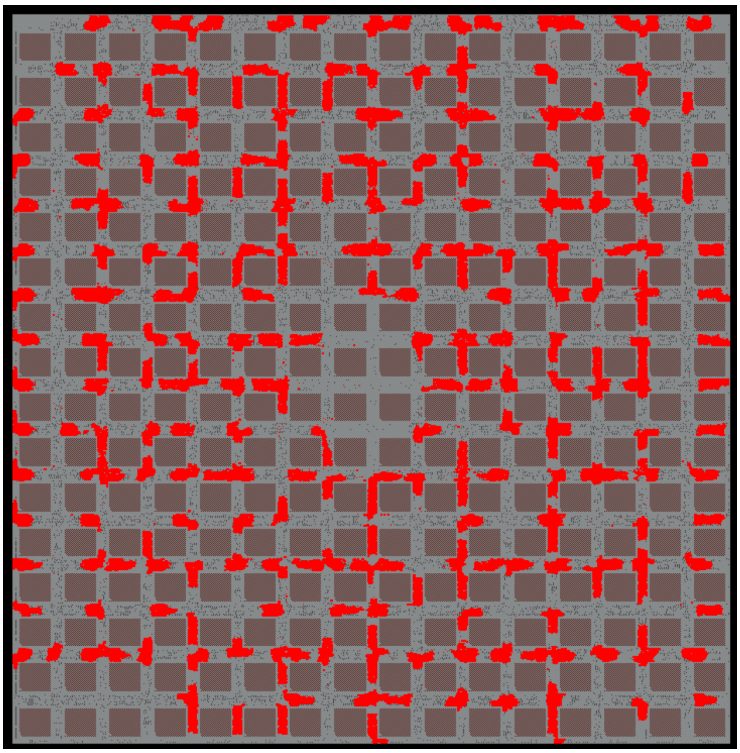

Figure 12: Four-input MAC (in red) are distributed in the pixelated area and placed in the proximity of the associated neighboring pixels.

multiplier as adders thanks to the fixed-point arithmetic.

Table 1 reports the latency and area estimate for the AE and PCA designs from Catapult HLS. The solutions highlighted have been integrated in the AI-In-Pixel-65 chip. It is worth noting that the modular solution provides the best area results for the AE design; in contrast, the PCA design has the best results with a function-inlining approach because of the sparsity and inherent logic simplification.

7. Area overhead in the digital implementation

In order to measure the overhead caused by the compression algorithms for the digital implementation in the 65nm Low Power CMOS process, we use a baseline implementation that lacks the compression logic. The area occupied by

Table 1: Latency and area estimate for the AE and PCA HLS designs. Latency is in clock cycles; area is in mm^2.

	AE		PCA	
HLS Solution	Latency	Area	Latency	Area
modular	30	0.549	30	1.516
inlined	1	1.700	1	0.652

the analog islands containing the pixels and by the digital logic for the baseline chip is $2,560,000$ μm^2; while the total area for the chips with the AE and the PCA algorithms is $3,097,600$ μm^2 in both instances. Thus, the compression algorithm caused an overhead of 21% in the total area.

8. Conclusions

The AI-In-Pixel-65 design delivers a low-noise, low-power, compact solution for the readout of pixelated photodiode arrays. Two algorithms were implemented for lossy data compression, a PCA based compression algorithm and a machine learning autoencoder, to reduce the data required to be transferred off-chip. The algorithms were designed such that they can be implemented and computed directly within the pixel area, without the need for first transferring data to the periphery of the pixel array. The PCA and AE algorithms are capable of achieving 50× and 70× compression of the data, while still maintaining sufficient image quality for use in reconstruction of X-ray ptychography measurements.

References

[1] C. Jacobsen, J. Deng, Y. Nashed, Strategies for high-throughput focused-beam ptychography, Journal of Synchrotron Radiation 24 (5) (2017) 1078–1081.

[2] M. Du, Z. Di, D. Gürsoy, R. P. Xian, Y. Kozorovitskiy, C. Jacobsen, Upscaling x-ray nanoimaging to macroscopic specimens, Journal of applied crystallography 54 (2) (2021) 386–401.

[3] F. Fahim, S. Bianconi, J. Rabinowitz, S. Joshi, H. Mohseni, Dynamically reconfigurable data readout of pixel detectors for automatic synchronization with data acquisition systems, Sensors 20 (9) (2020) 2560.

[4] G. A. Carini, et al., Hybridized maps with an in-pixel a-to-d conversion readout asic, Nuclear Instruments and Methods in Physics Research 935 (2019) 232–238. doi:https://doi.org/10.1016/j.nima.2019.03.091.
URL https://www.sciencedirect.com/science/article/pii/S0168900219304334

[5] F. Fahim, et al., A low-power, high-speed readout for pixel detectors based on an arbitration tree, IEEE Trans. VLSI Syst. 28 (2) (2020) 576–584. doi:10.1109/TVLSI.2019.2953871.
URL https://doi.org/10.1109/TVLSI.2019.2953871

[6] P. Moreira, The LpGBT project status and overview, in: Fifth Common Atlas CMS Electronics Workshop for LHC Upgrades (ACES), 2016.

[7] S. Kulis, et al., The lpGBTv1 manual, https://lpgbt.web.cern.ch/lpgbt/ (2020).

[8] H. Li, Z. Xuan, A. Titriku, C. Li, K. Yu, B. Wang, A. Shafik, N. Qi, Y. Liu, R. Ding, et al., A 25 Gb/s, 4.4 V-swing, AC-coupled ring modulator-based WDM transmitter with wavelength stabilization in 65nm CMOS, IEEE Journal of Solid-State Circuits 50 (12) (2015) 3145–3159.

[9] A. Quinn, M. B. Valentin, T. Zimmerman, D. Braga, S. Memik, F. Fahim, A Cryogenic Readout IC with 100 KSPS in-Pixel ADC for Skipper CCD-in-CMOS Sensors, IEEE International Symposium on Circuits and Systems (ISCAS) (2023). doi:10.48550/arXiv.2302.03727.

[10] H. Wey, W. Guggenbuhl, An improved correlated double sampling circuit for low noise charge coupled devices, IEEE transactions on circuits and systems 37 (12) (1990) 1559–1565.

[11] T. Zimmerman, G. Deptuch, F. Fahim, Compact, low power, high resolution adc per pixel for large area pixel detectors., U.S. Patent 11,108,981, issued August 31, 2021.

[12] M. Hammer, K. Yoshii, A. Miceli, Strategies for on-chip digital data compression for x-ray pixel detectors, Journal of Instrumentation 16 (01) (2021) P01025. doi:10.1088/1748-0221/16/01/P01025.
URL https://dx.doi.org/10.1088/1748-0221/16/01/P01025

[13] P. Huang, M. Du, M. Hammer, A. Miceli, C. Jacobsen, Fast digital lossy compression for X-ray ptychographic data, Journal of Synchrotron Radiation 28 (1) (2021) 292–300. doi:10.1107/S1600577520013326.
URL https://doi.org/10.1107/S1600577520013326

[14] K. Wakonig, H.-C. Stadler, M. Odstrčil, E. H. R. Tsai, A. Diaz, M. Holler, I. Usov, J. Raabe, A. Menzel, M. Guizar-Sicairos, *PtychoShelves*, a versatile high-level framework for high-performance analysis of ptychographic data, Journal of Applied Crystallography 53 (2) (2020) 574–586. doi:10.1107/S1600576720001776.
URL https://doi.org/10.1107/S1600576720001776

[15] L. Loetgering, M. Rose, D. Treffer, I. A. Vartanyants, A. Rosenhahn, T. Wilhein, Data compression strategies for ptychographic diffraction imaging, Advanced Optical Technologies 6 (6) (2017) 475–483 [cited 2023-05-13]. doi:doi:10.1515/aot-2017-0053.
URL https://doi.org/10.1515/aot-2017-0053

[16] L. Loetgering, D. Treffer, T. Wilhein, Compression and information recovery in ptychography, Journal of Instrumentation 13 (04) (2018) C04019. doi:10.1088/1748-0221/13/04/C04019.
URL https://dx.doi.org/10.1088/1748-0221/13/04/C04019

[17] M. J. Cherukara, T. Zhou, Y. Nashed, P. Enfedaque, A. Hexemer, R. J. Harder, M. V. Holt, AI-enabled high-resolution scanning coherent diffraction imaging, Applied Physics Letters 117 (4),

044103 (07 2020). arXiv:https://pubs.aip.org/aip/apl/article-pdf/doi/10.1063/5.0013065/13188837/044103_1_online.pdf, doi:10.1063/5.0013065.
URL https://doi.org/10.1063/5.0013065

[18] H. Chan, Y. S. G. Nashed, S. Kandel, S. O. Hruszkewycz, S. K. R. S. Sankaranarayanan, R. J. Harder, M. J. Cherukara, Rapid 3D nanoscale coherent imaging via physics-aware deep learning, Applied Physics Reviews 8 (2), 021407 (05 2021). arXiv:https://pubs.aip.org/aip/apr/article-pdf/doi/10.1063/5.0031486/14581218/021407_1_online.pdf, doi:10.1063/5.0031486.
URL https://doi.org/10.1063/5.0031486

[19] M. Van Heel, M. Schatz, Fourier shell correlation threshold criteria, Journal of structural biology 151 (3) (2005) 250–262.

[20] P. Huang, Toward large-scale x-ray microscopy for ptychography and fluorescence tomography, Ph.D. thesis, Northwestern University (2022).

[21] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, T. Aarrestad, V. Loncar, J. Ngadiuba, M. Pierini, A. A. Pol, S. Summers, Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors (2020).
URL https://arxiv.org/abs/2006.10159

[22] S. Lahti, P. Sjövall, J. Vanne, T. D. Hämäläinen, Are we there yet? a study on the state of high-level synthesis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38 (5) (2018) 898–911.

[23] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, et al., A survey and evaluation of fpga high-level synthesis tools, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35 (10) (2015) 1591–1604.

[24] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. D. Guglielmo, P. C. Harris, J. D. Krupa, D. S. Rankin, M. B. Valentin, J. D. Hester, Y. Luo, J. Mamish, S. Ogrenci-Memik, T. Aarrestad, H. Javed, V. Loncar, M. Pierini, A. A. Pol, S. Summers, J. M. Duarte, S. Hauck, S. Hsu, J. Ngadiuba, M. Liu, D. Hoang, E. Kreinar, Z. Wu, hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices, CoRR abs/2103.05579 (2021). arXiv:2103.05579. URL `https://arxiv.org/abs/2103.05579`

[25] Mentor/Siemens, HLSLibs: Open-Source High-Level Synthesis IP Libraries, `https://hlslibs.org` (2020).