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

Danny Noonan [1], Davide Braga [1], Giuseppe Di Guglielmo [1,2], Priyanka Dilip [1], Farah Fahim [1,2], Panpan Huang [2], Chris Jacobsen [2], Seda Ogrenci [2], Adam Quinn [2], Nhan Tran [1,2], Manuel B. Valentin [2], Thomas Zimmerman [1]

[1] Fermilab

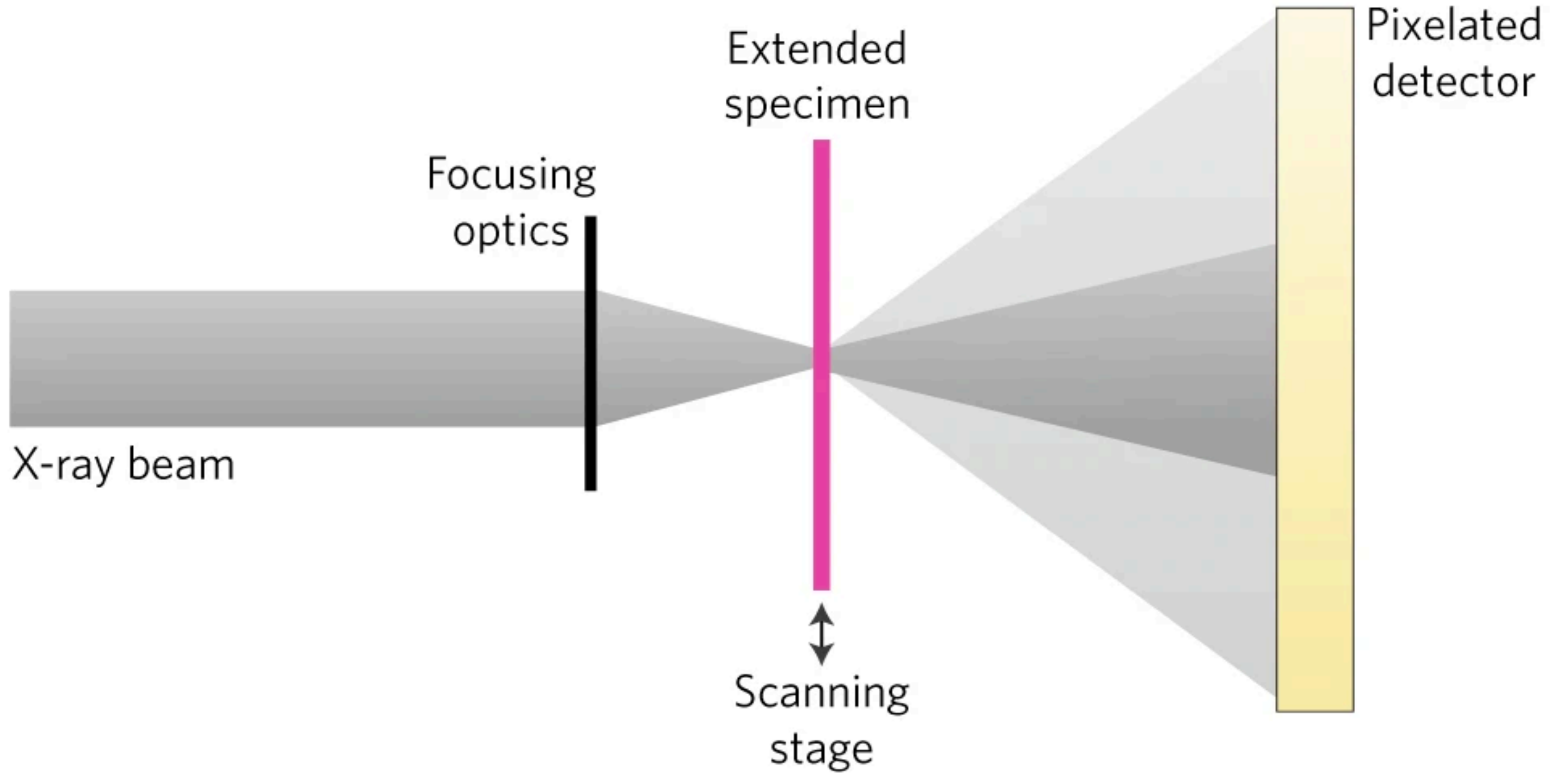
[2] Northwestern University

TWEPP 2023
October 1-6, 2023

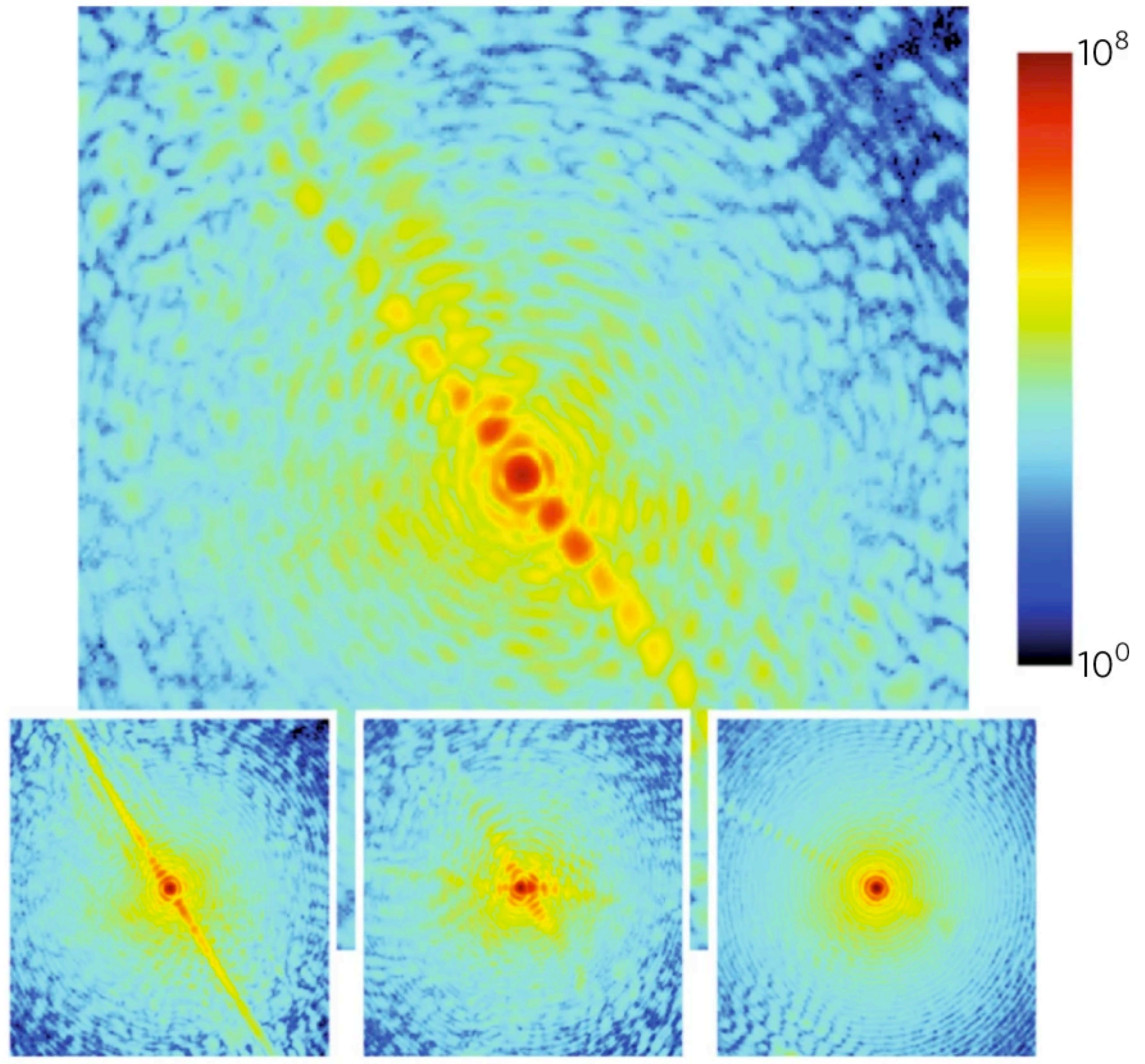
Introduction

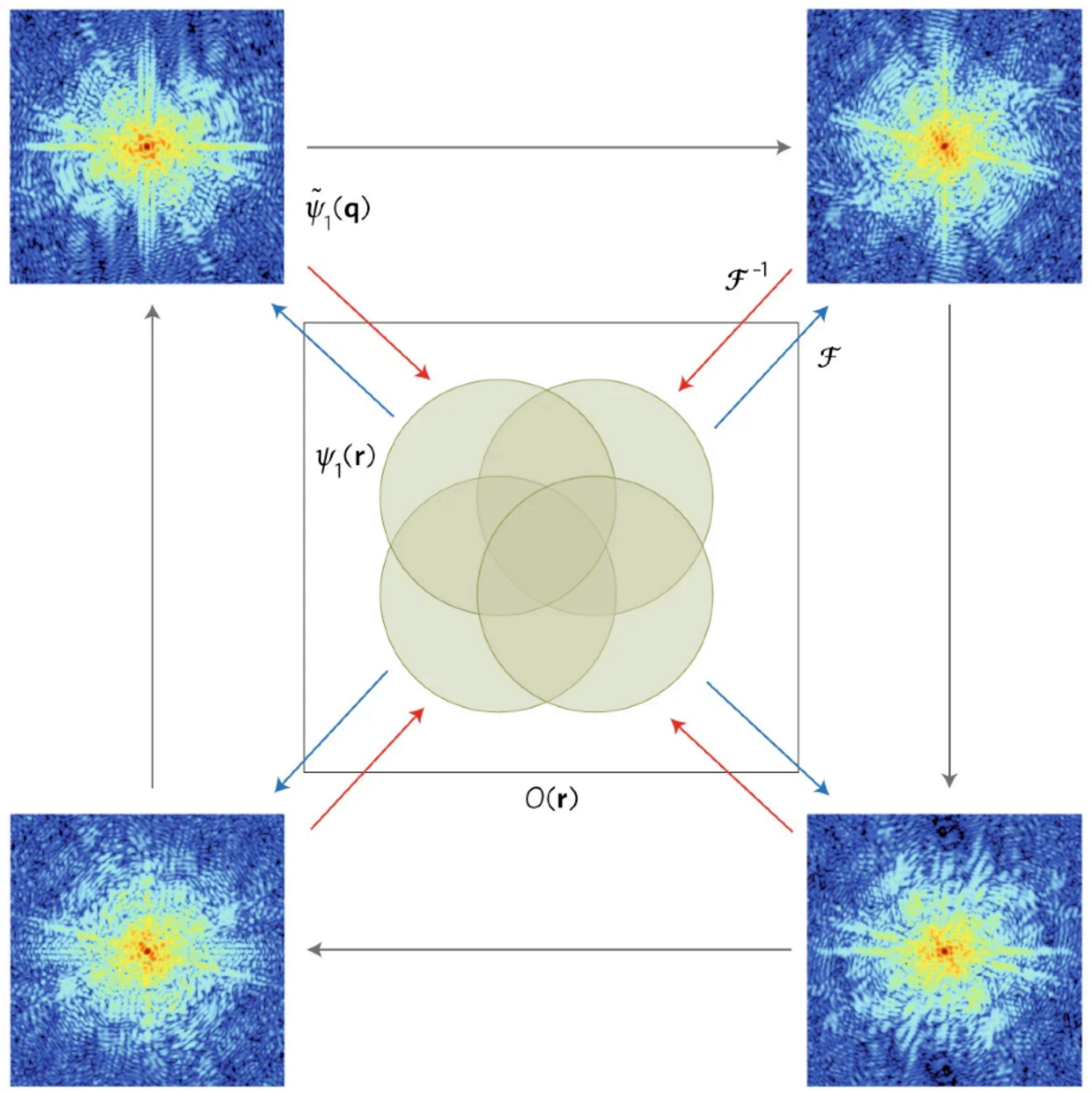
- Fast frame-rate, higher resolution detectors are essential for improving performance of X-ray microscopy techniques
- Pixelated front-end readouts are hitting data I/O bottleneck
  - Single 400x400 pixel chip, with 10-bit ADC, operating a 1 Mfps, generates 1.6 Tbps of data
  - Frame rates become limited not by time to integrate and digitize data, but by off-chip data transfer
- For operating without dead-time, data transfer needs to occur at same rate as digitization
  - Transferring O(Tbps) off detector is not feasible
  - Need data reduction on-chip
- **AI-In-Pixel-65**
  - Test chip for pixelated read out of X-ray detectors (specifically targetting X-ray ptychography)
  - Capable of 50-70x lossy data compression
  - Data compression performed within pixel area rather than chip periphery

X-ray Ptychography



- X-ray microscopy technique
  - Computationally reconstruct image of a specimen
  - Diffraction patterns collected over scan of specimen
- Collects large amounts of data
  - Sampled in overlapping positions
- Redundancy in data lends itself to be suitable for data compression
  - Compression techniques already common for off-detector storage



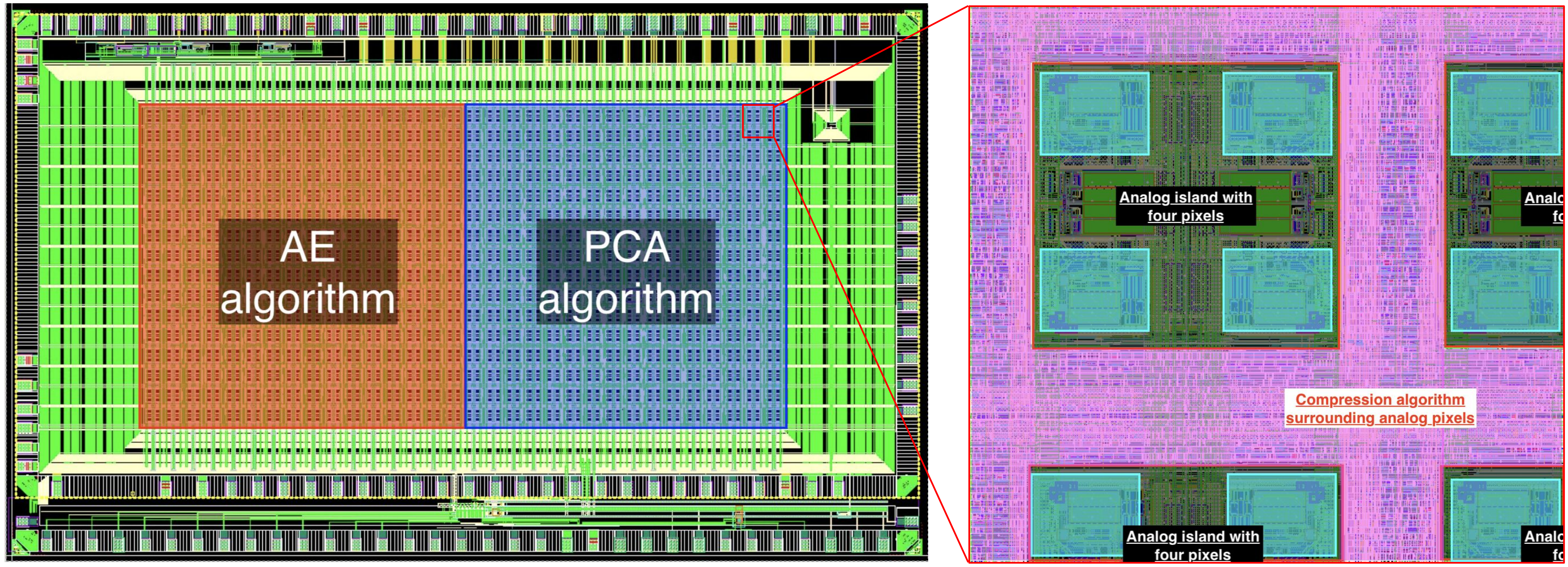


Pfeiffer, F. X-ray ptychography. *Nature Photon* **12**, 9–17 (2018). https://doi.org/10.1038/s41566-017-0072-5

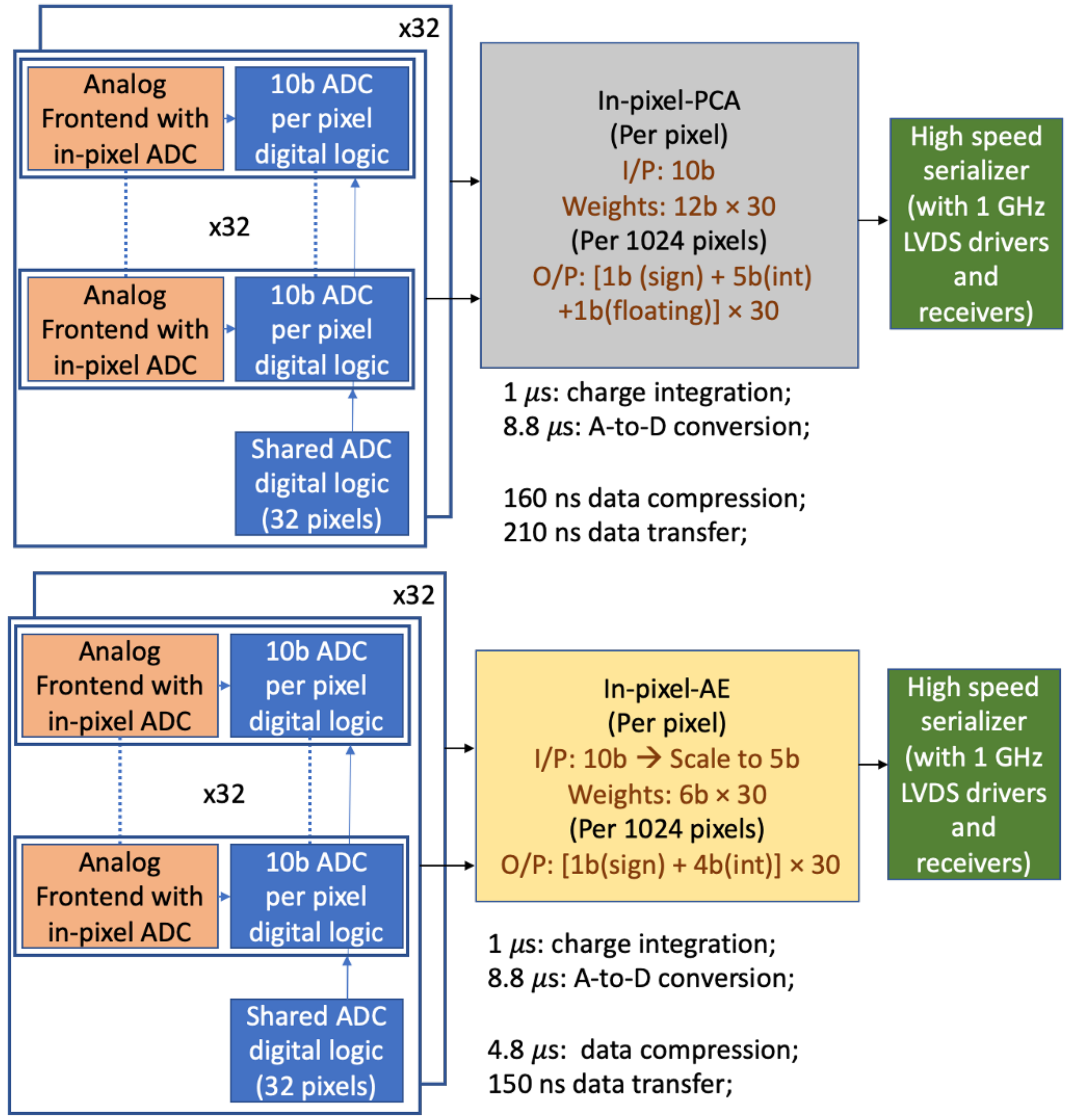
Data reduction

- Reducing data volumes as close to the source as possible removes the data transfer bottleneck
  - Integrating data reduction schemes into Read-Out Integrated Circuits (ROIC)
- Various options available for data reduction at source:
  - Data sparsification: zero suppression of data
    - Potentially high overhead (16-bit addressing per pixel for 200x200 pixel array)
    - May not provide much gain in noisy data (~60% zero pixels)
  - Data compression:
    - Principal Component Analysis (PCA)
    - Machine learning based data compression through an AutoEncoder

AI-In-Pixel-65



- ROIC test chip
  - Designed in 65nm Low Power CMOS
- Pixelated readout for X-ray detectors
- Pair of 32-by-32 pixel arrays with independent data compression algorithms
  - Signals digitized by 10-bit SAR ADC at 100k samples per second
  - Data-compression implemented in-pixel (rather than at periphery)
  - Pixel area of 55x55 μm² with data compression implented (expanded from 50x50 μm² without compression)
- Two algorithms for data compression in each of two halves
  - AutoEncoder
  - Principal Component Analysis



Principal Component Analysis (PCA)

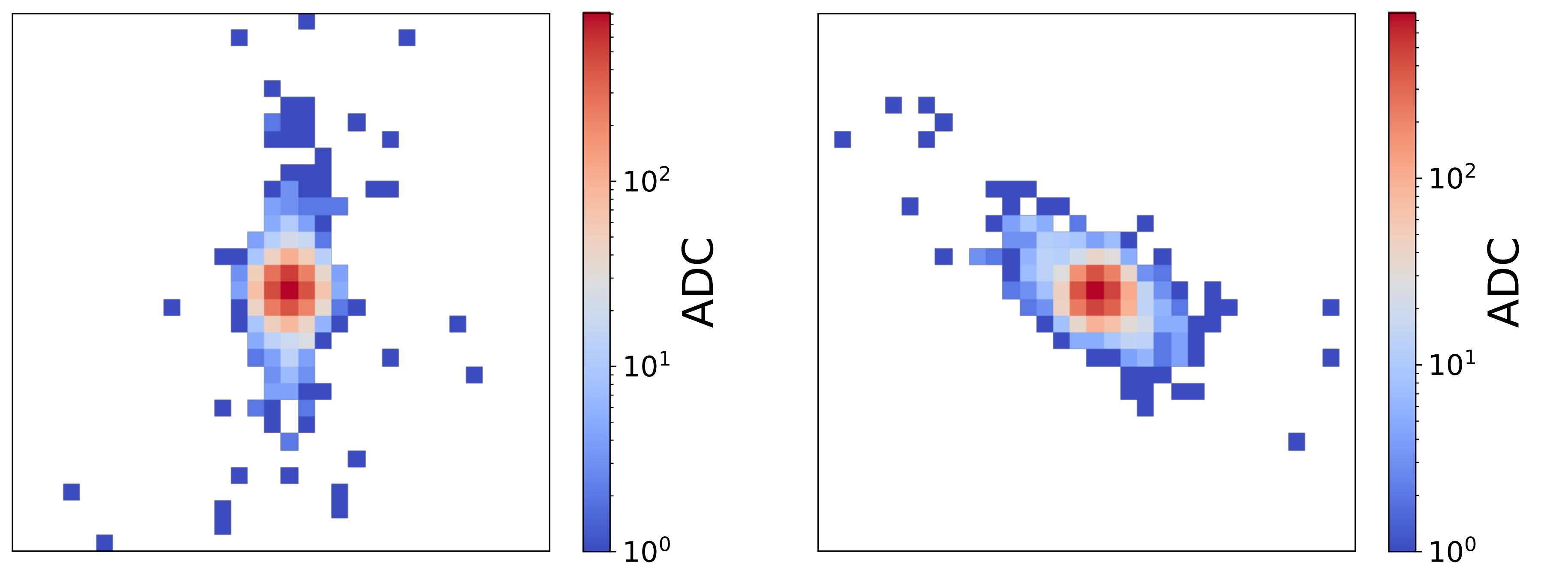
- Represent diffraction patterns as a linear combination of features
  - Diffraction pattern from array of j pixels (D_j) can be represented as product of eigenimages ($R_{n\times j}$) and the corresponding eigenvalues (P_n)

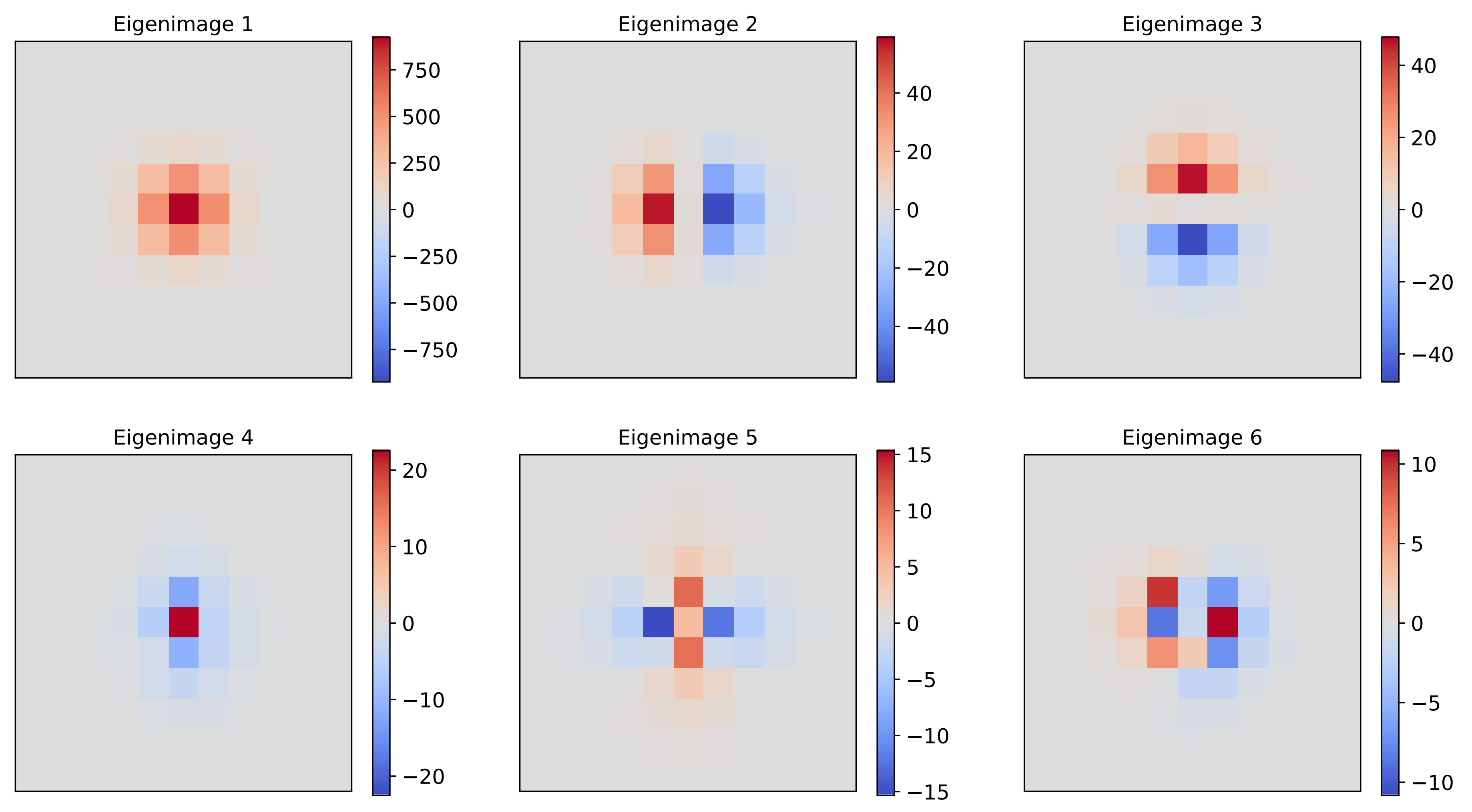
$$D_j = P_n \times R_{n\times j}$$

- Eigenvalues can be calcuated inverting eigenimage matrix ($R^{-1}_{j\times n}$)

$$P_n = D_j \times R^{-1}_{j\times n}$$

- R^{-1} can be used to calculate eigenvalues on chip
  - Since n << j, reading out eigenvalues instead of full array of pixels reduces data
  - Varying the number of eigenvalues used varies level of compression

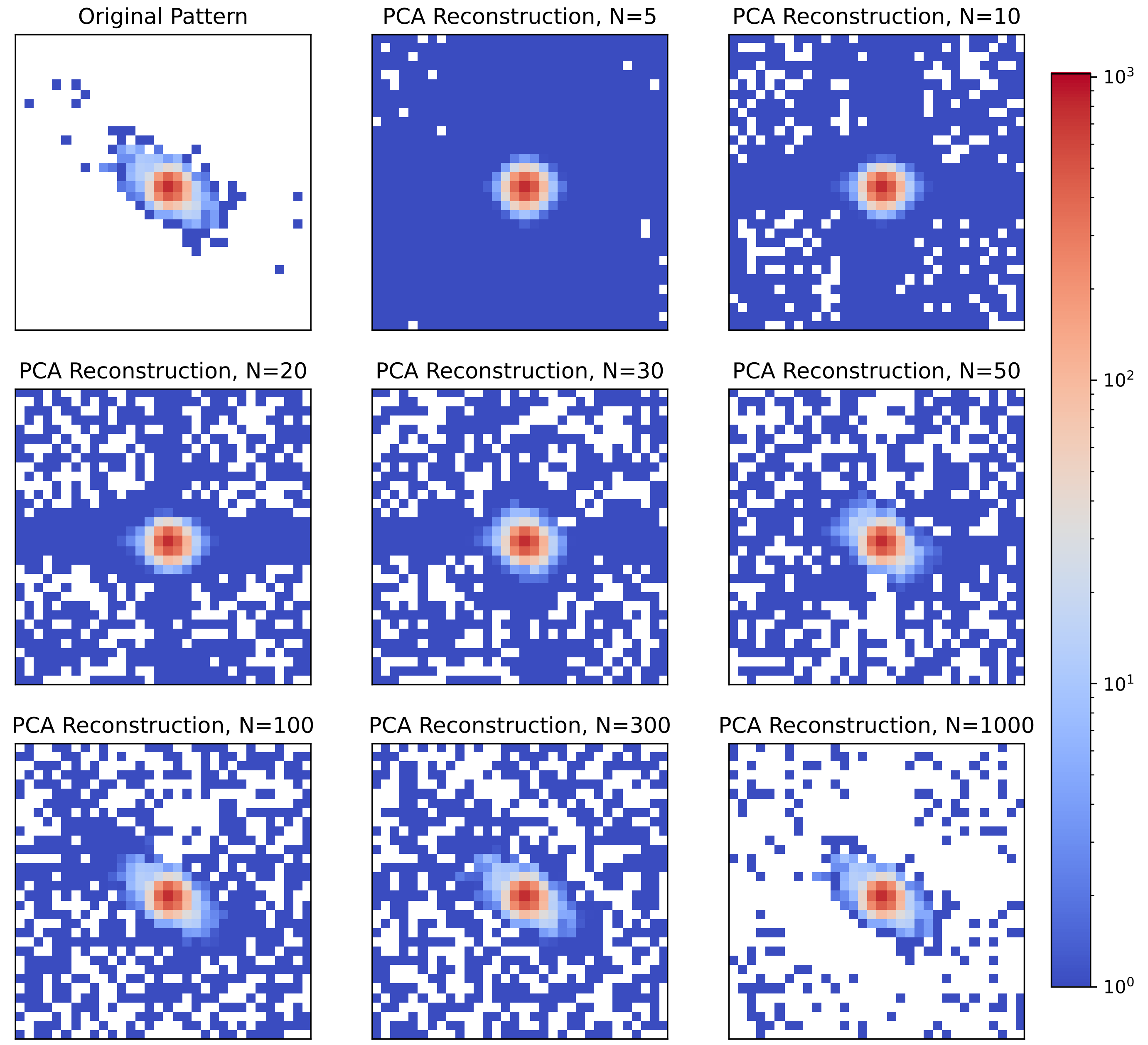


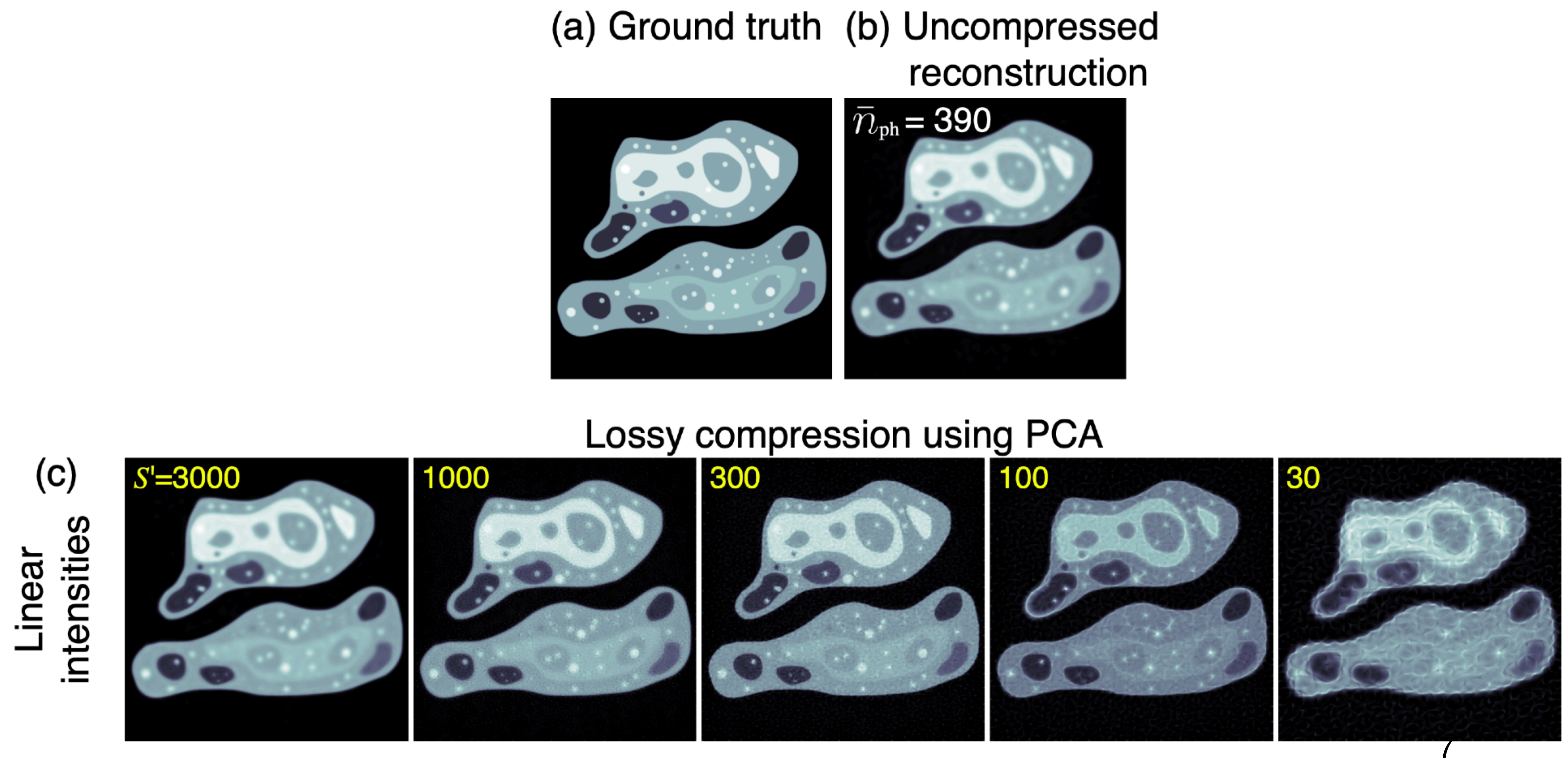


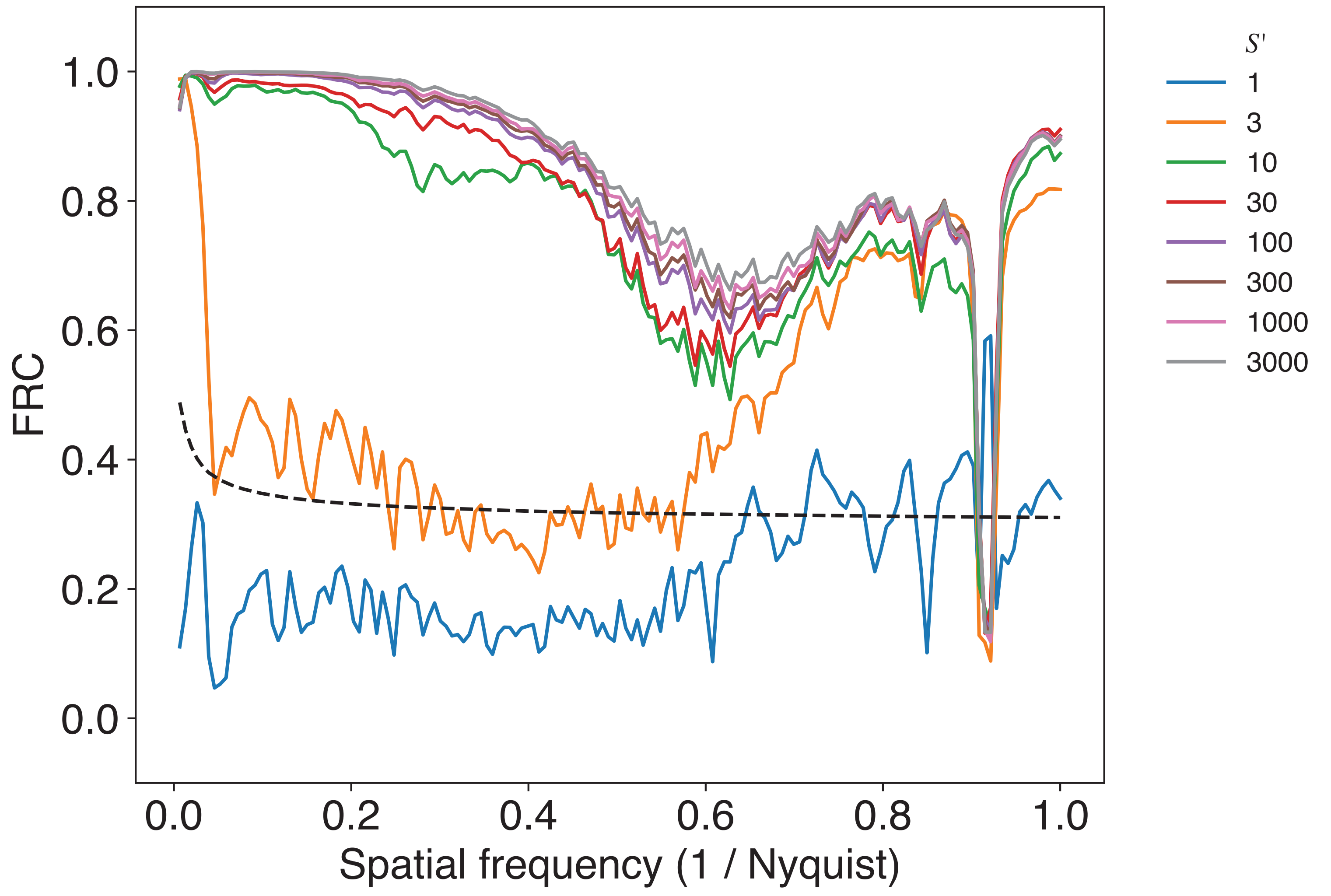
PCA

- Varying the number of eigenvalue/eigenimages effects the quality of the diffraction patterns
- Compute Forrier Ring Coefficient (FRC) as a metric for quality of reconstruction images
  - FRC compares similarity of two images (reconstructed vs original) at varying spatial resolutions
  - Optimize number of outputs and precision of weights and outputs based on quality of image reconstruction
  - With 30 eigenvalues, can maintain quality of sampled image
- With 30 eigenvalues at 7-bit precision, achieve 50x compression (1024 x 10-bit to 32 x 7-bit)

Diffraction patters, reconstructed with different compression levels



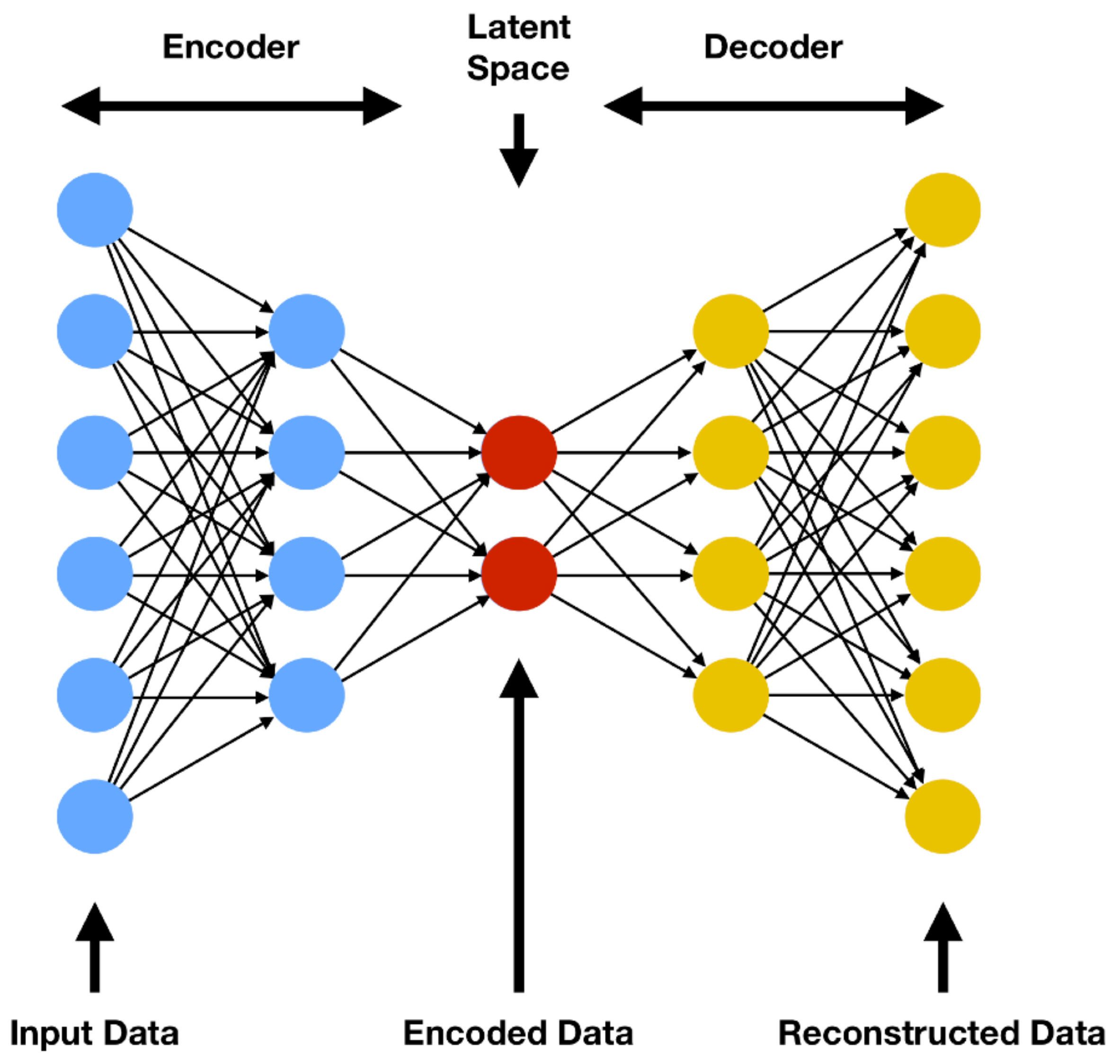


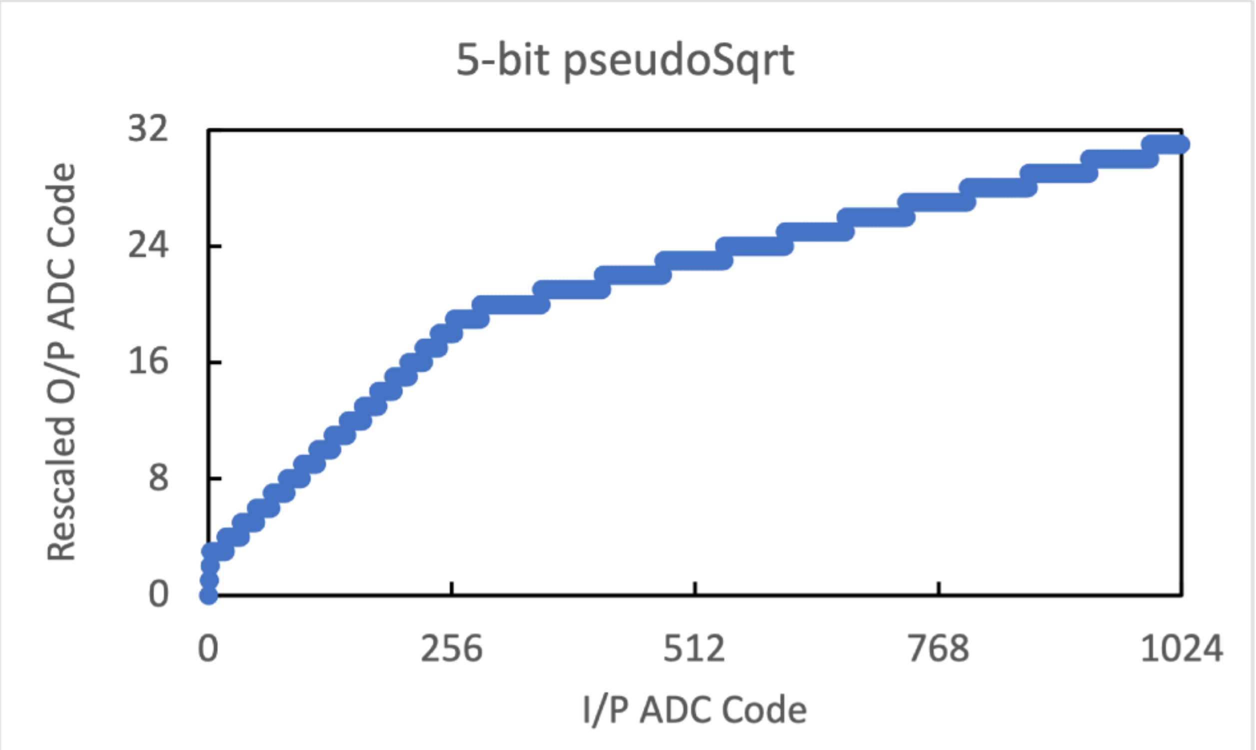


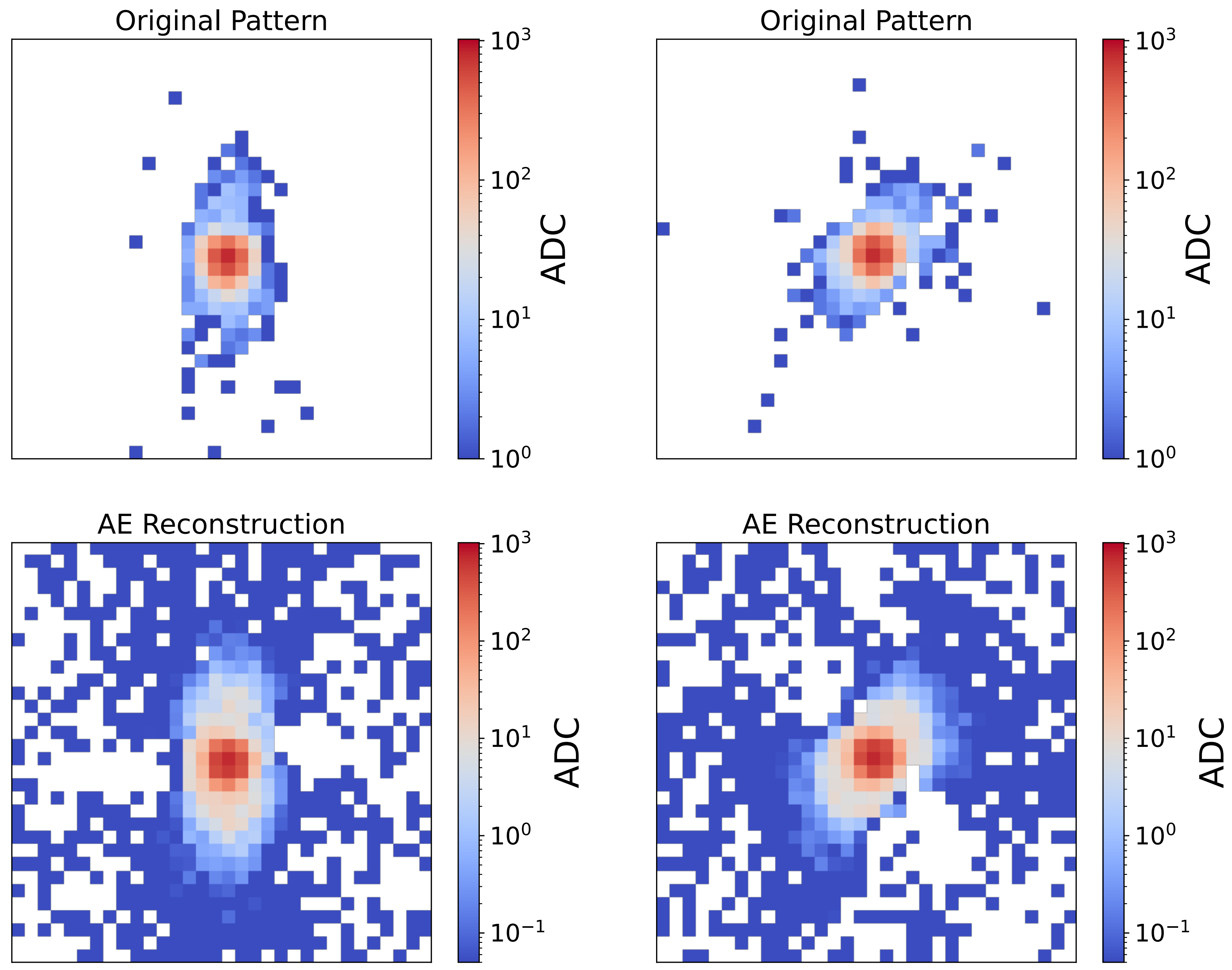
AutoEncoder

- PCA matrix multiplication is essentially just a neural network dense layer
  - Can we do the same, but using machine learning to determing the weights?
- AE algorithm, uses same basic structure as PCA
  - Fully connected dense layer
  - Maintains 30-value latent space
  - Trains encoder and decoder network simultaneously, compressing and decompressing the image from the latent space
- Preprocessing of data from 10-bit ADC value into a 5-bit approximation of square root
- Quantization aware training using QKeras
  - Network can learn how to best make use of available precision
- 70x compression factor: 30 latent space values at 5-bit bit precision

Example AutoEncoder Network Structure

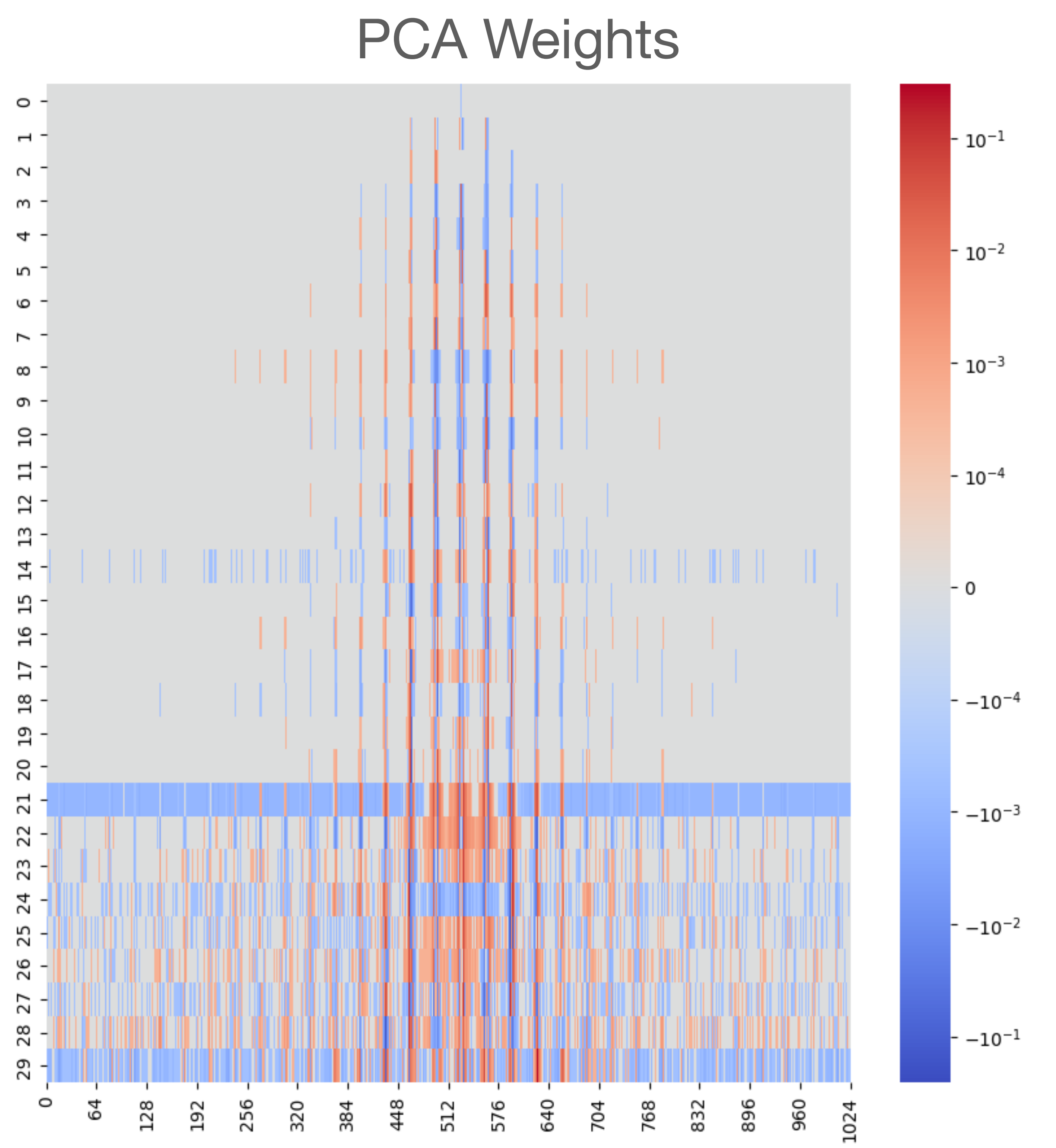


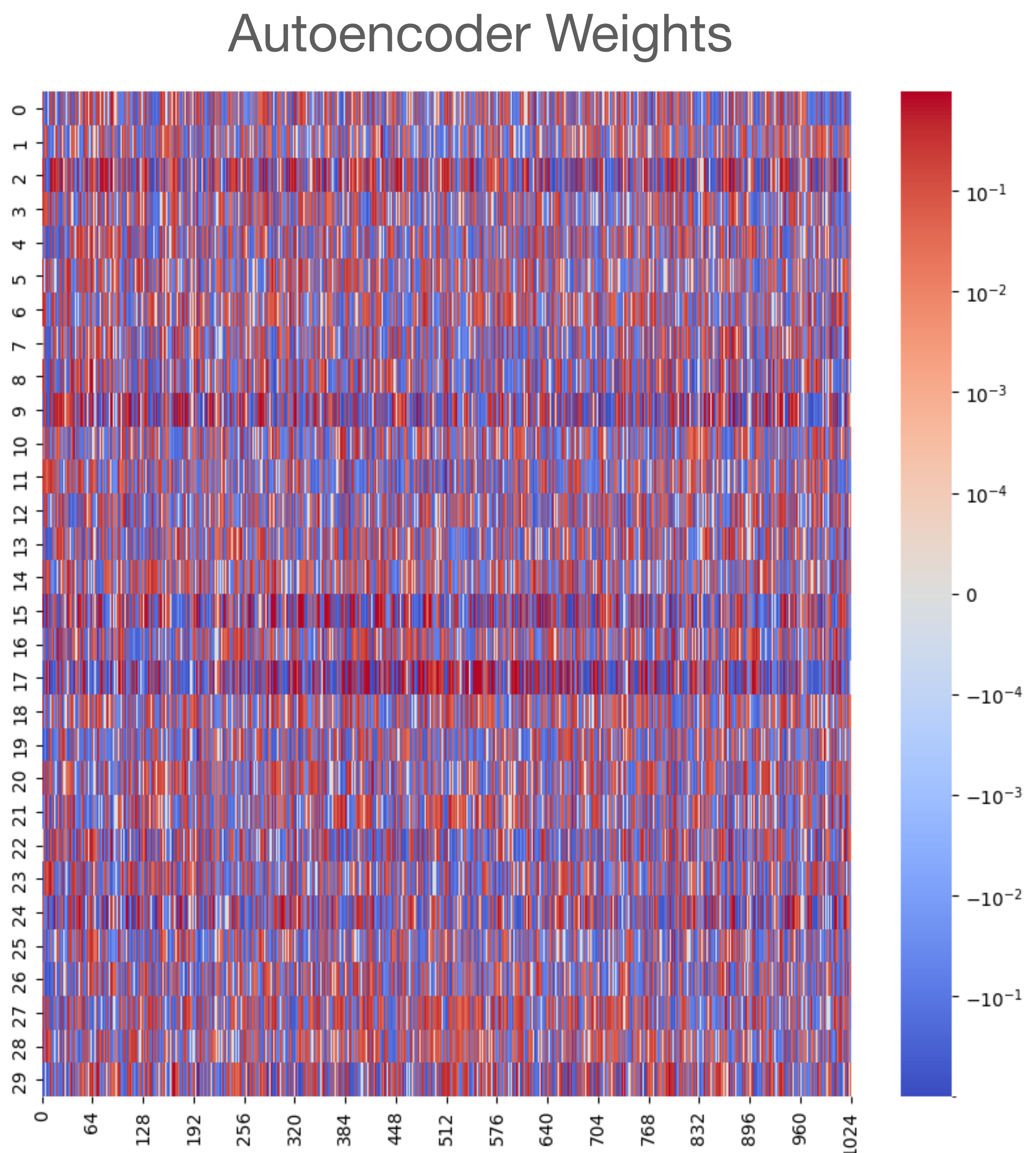




Algorithm Weight Comparison

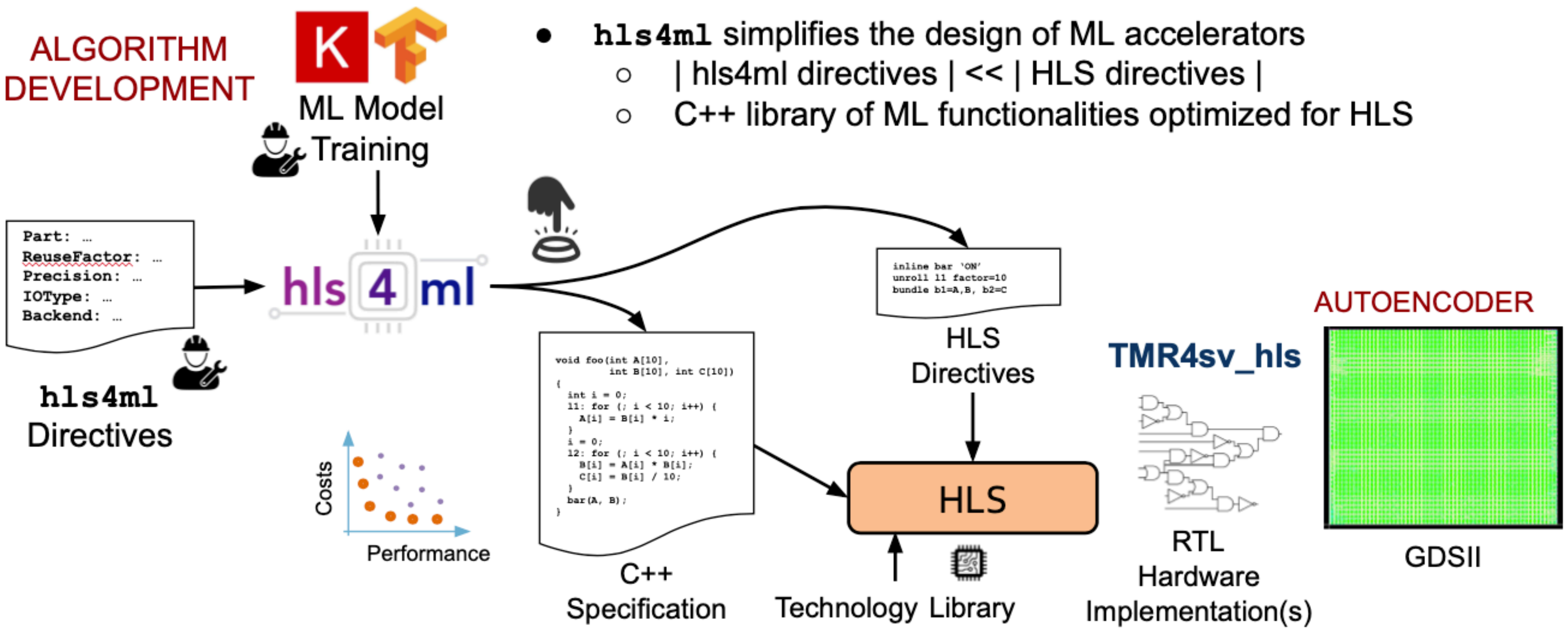
- PCA and AE algorithms have same number of weights, but very different scales and precision
  - 30 x 1024 weights
- PCA:
  - Requires 12-bit precision
  - 77.98% zero-valued weights
- AutoEncoder:
  - 6-bit precision in weights
  - 8.69% zero-valued
- Difference in distributions of weights leads to different implementation strategies





Design Methodology

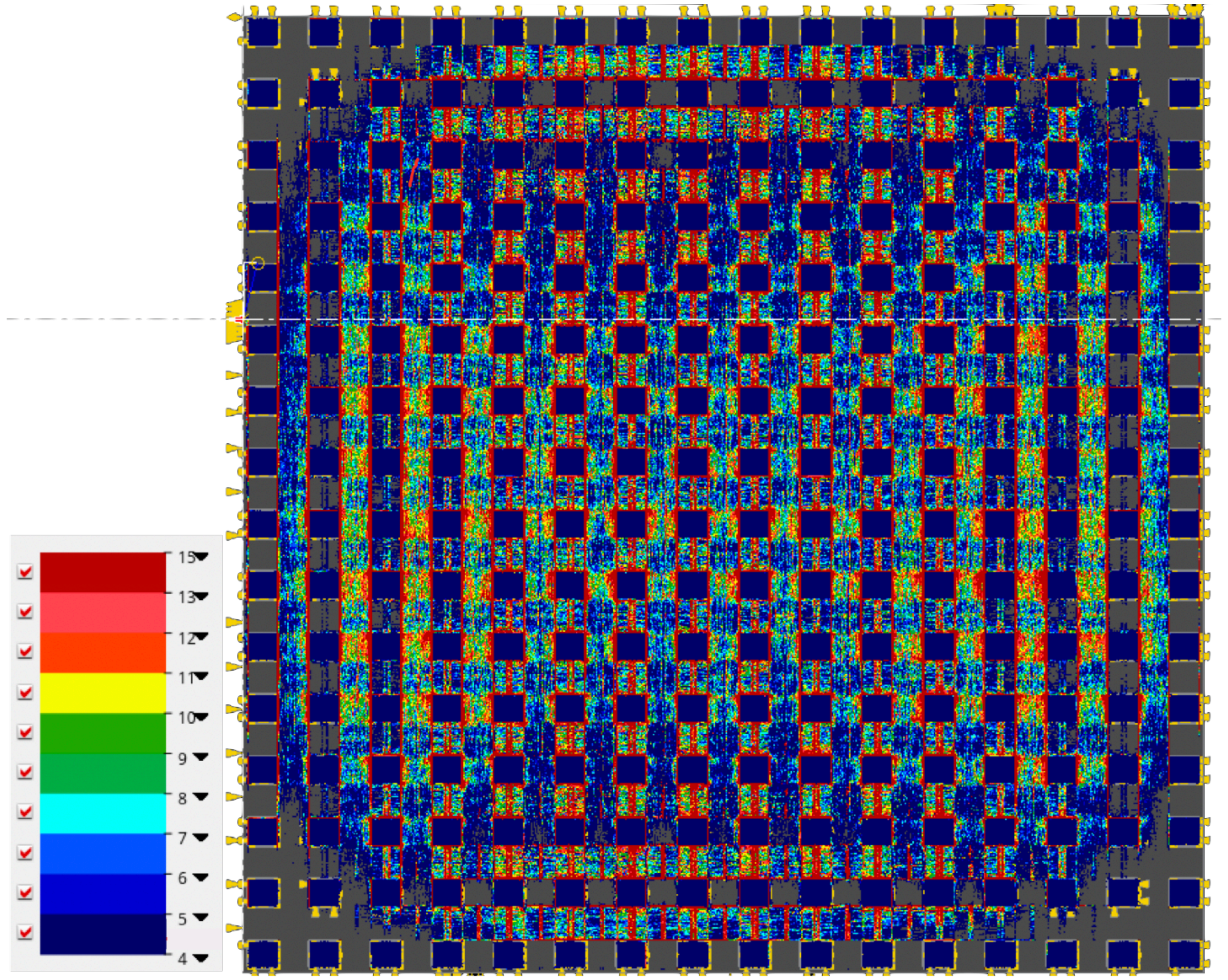
- Design goal of avoid moving data to periphery of ROIC
  - Implement calculations in-pixel
- Leverage HLS (Siemens Catapult HLS) and hls4ml for implementation

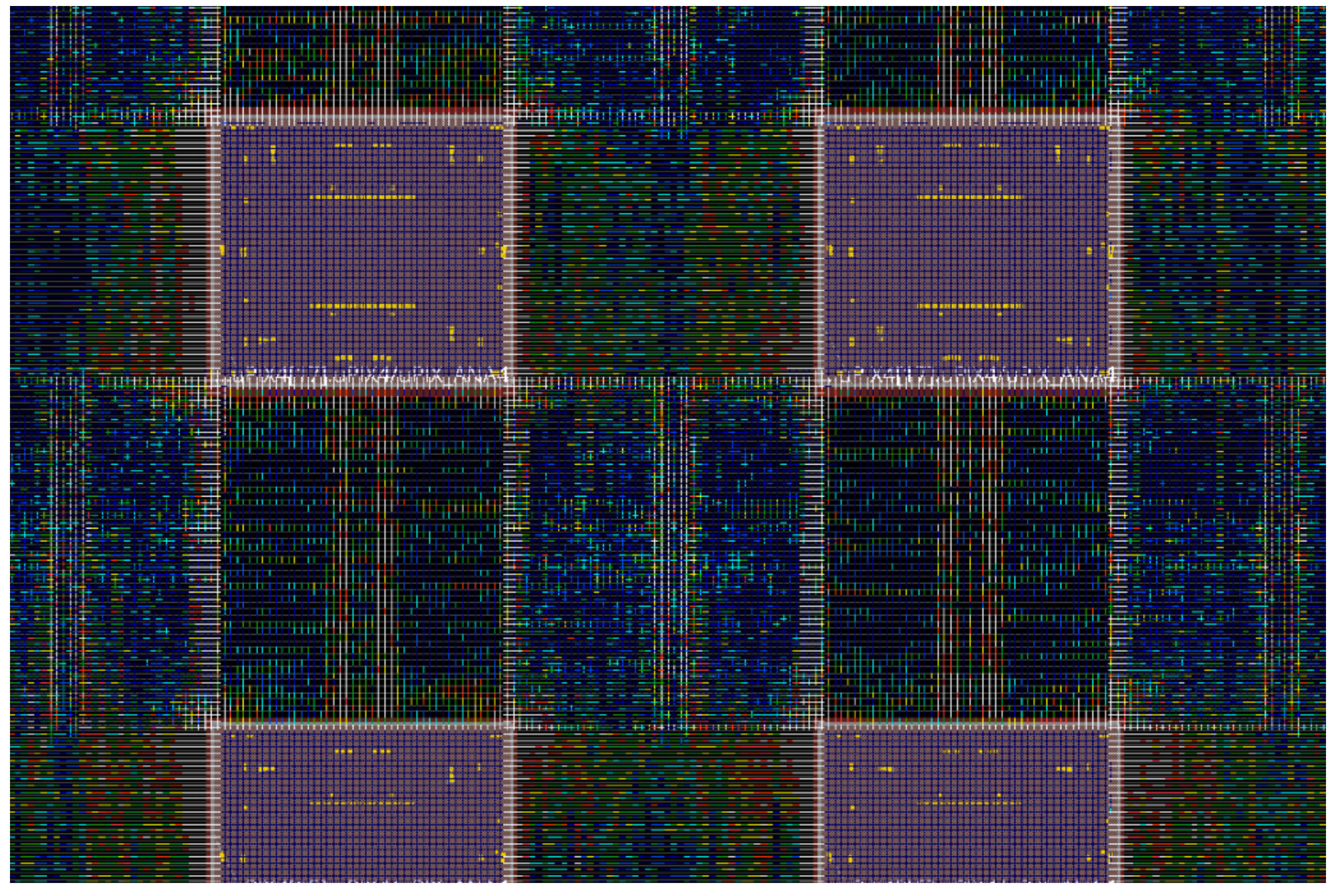


Implemenation Issues

- Fully dense architectures present challenges when with signal routing when performing the compression “in-pixel”
- Congestion issues with initial 50 μm x 50 μm pixel size
  - Increase to 55 μm pitch relieves some of the issues
- Changes to architecture choices in HLS to allow for easier place and route
  - Different strategies implemented for PCA and AE optimization

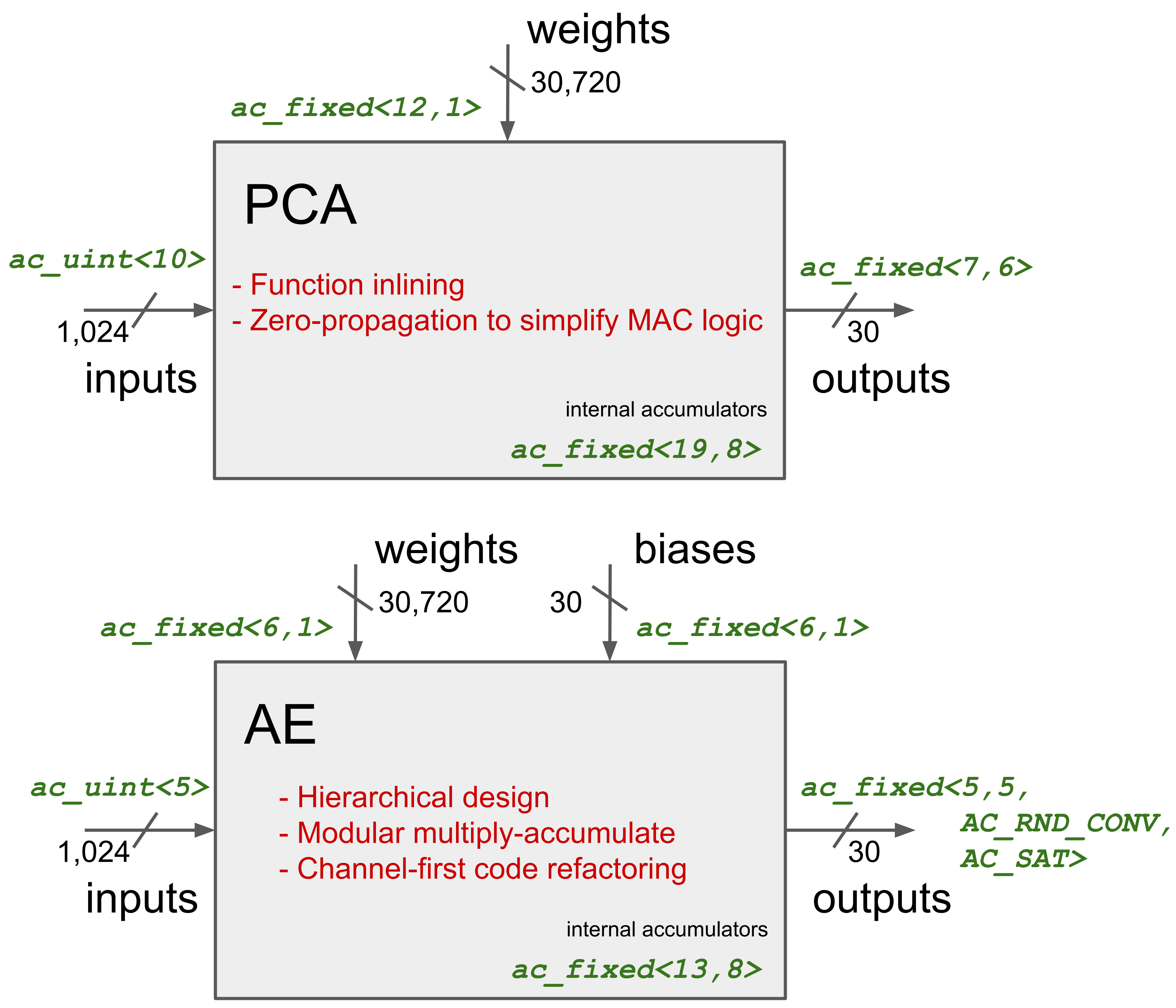
Routing congestion in AE implementation

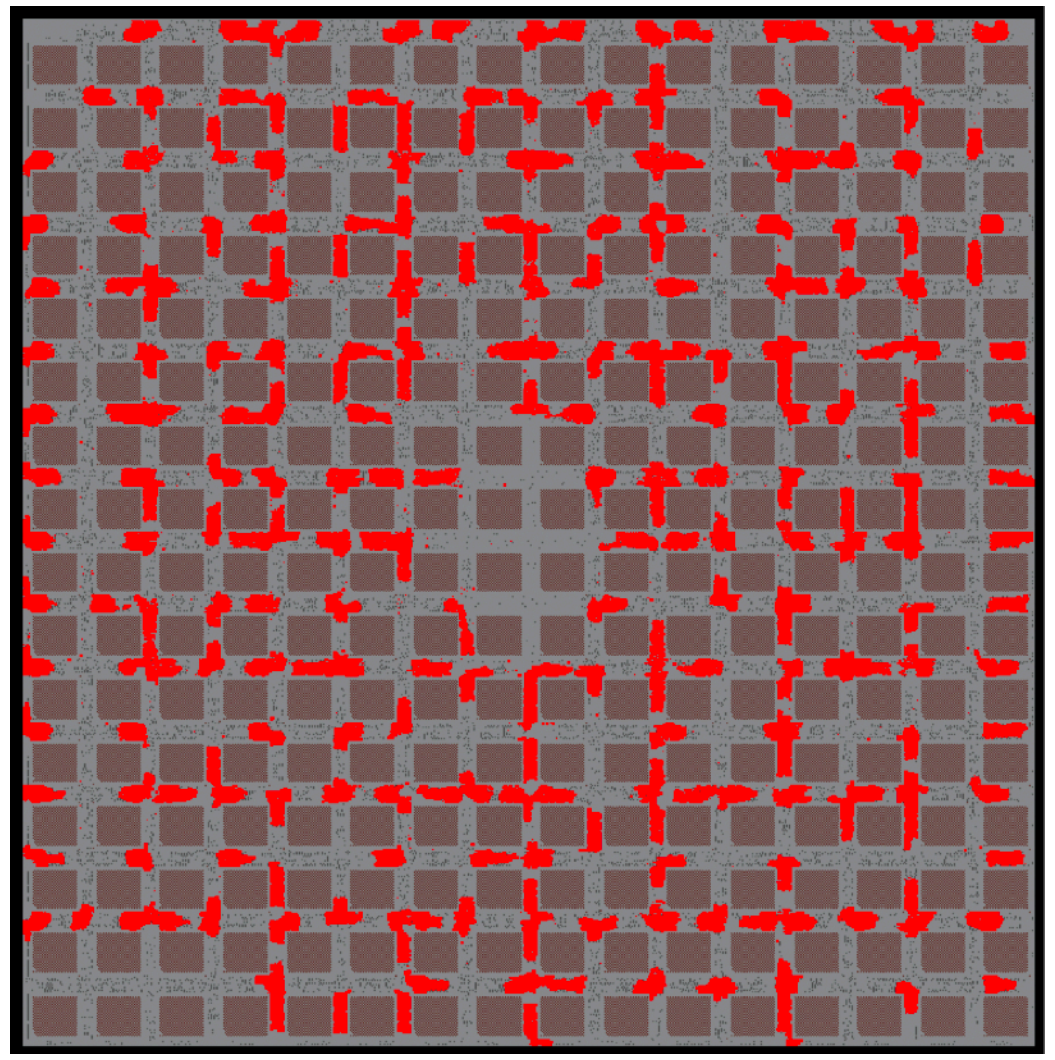




Congestion Improvements

- PCA:
  - Unroll loops
  - Take advantage of sparsity in weights to simplify accumulators (zero multiplications removed in synthesis)
- Autoencoder:
  - Refactor HLS code, allowing for module multiply-accumulate step, and pipelining the 30 calculations
  - HLS code modifications to perform accumates among 4 neighboring pixels first, reducing routing requirements
- With these changes, able to fit both compression into their respective pixelated area, with different latencies for the two algorithms





Distribution of MAC in pixelated area of AE

Area estimates for different HLS design choices

	AE		PCA	
	Latency	Area (mm2)	Latency	Area (mm2)
Modular	30	0.549	30	1.516
Inline	1	1.700	1	0.652

Summary

- AI-In-Pixel-65 demonstrates ability to perform data compression directly within a pixelated front end read out chip
- Two algorithms explored for data compression,
  - **Principal Component Analysis**: achieves 50x compression, 1 clock cycle of latency
  - **AutoEncoder**: 70x data compression, 30 clock cycles
- Implementation strategies in HLS customized to needs of each algorithm
- These levels of compression would make larger and faster pixel arrays more feasible to readout
  - 400x400 pixels x 10b x 1 Mfps -> 1.6 Tbps data, becomes 32 Gbps with 50x compression

BACKUP

HLS optimization for better layout routing

```
#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];
  }
}
```

Initial “Channel Last” HLS implementation
1024 multiplier accumulators (MACs)

```
#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;
  }
}
```

“Channel First” Implementation
Separated into 256 four-input MACs