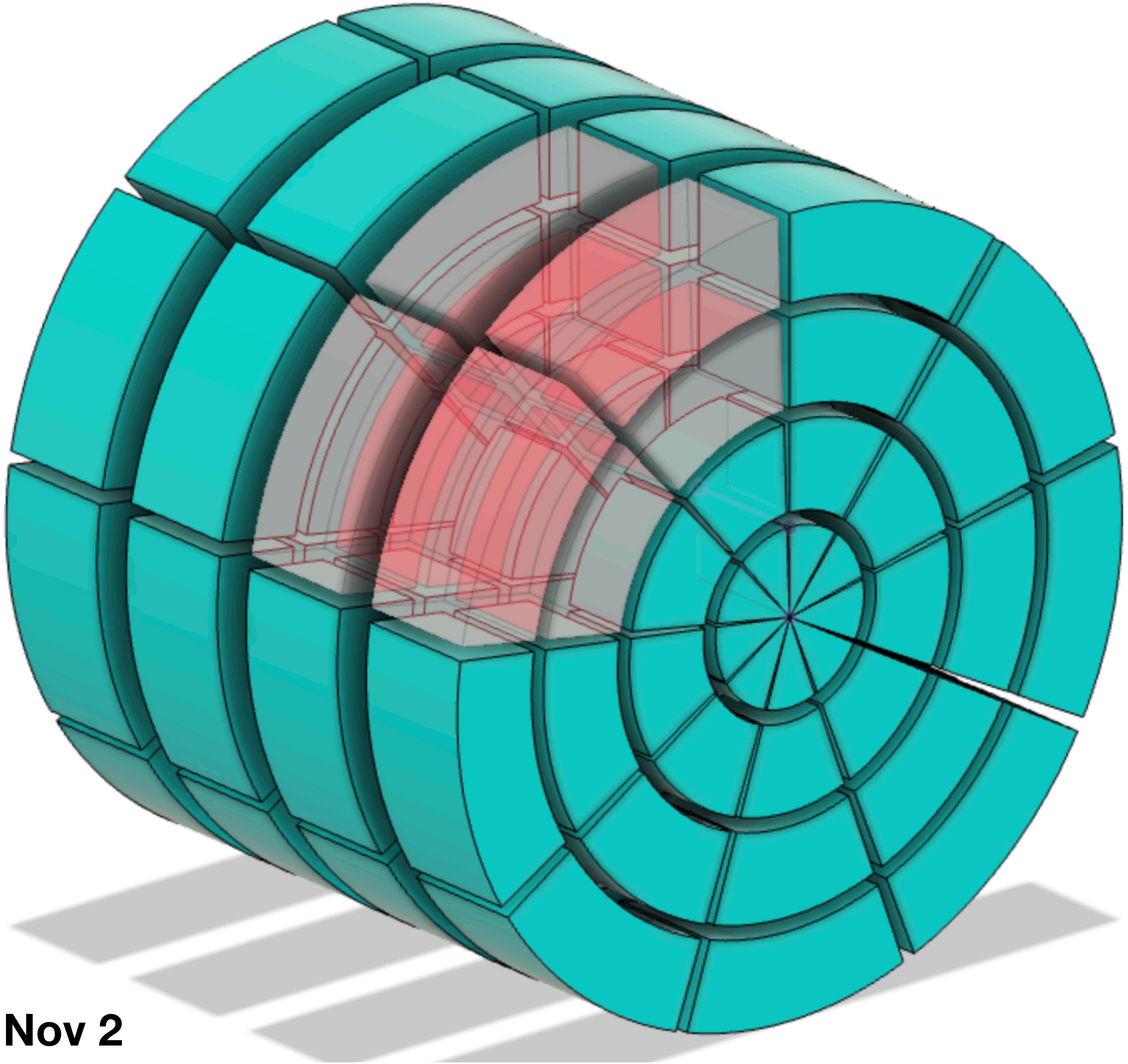
Fast Calorimeter Simulation with VQ-VAEs



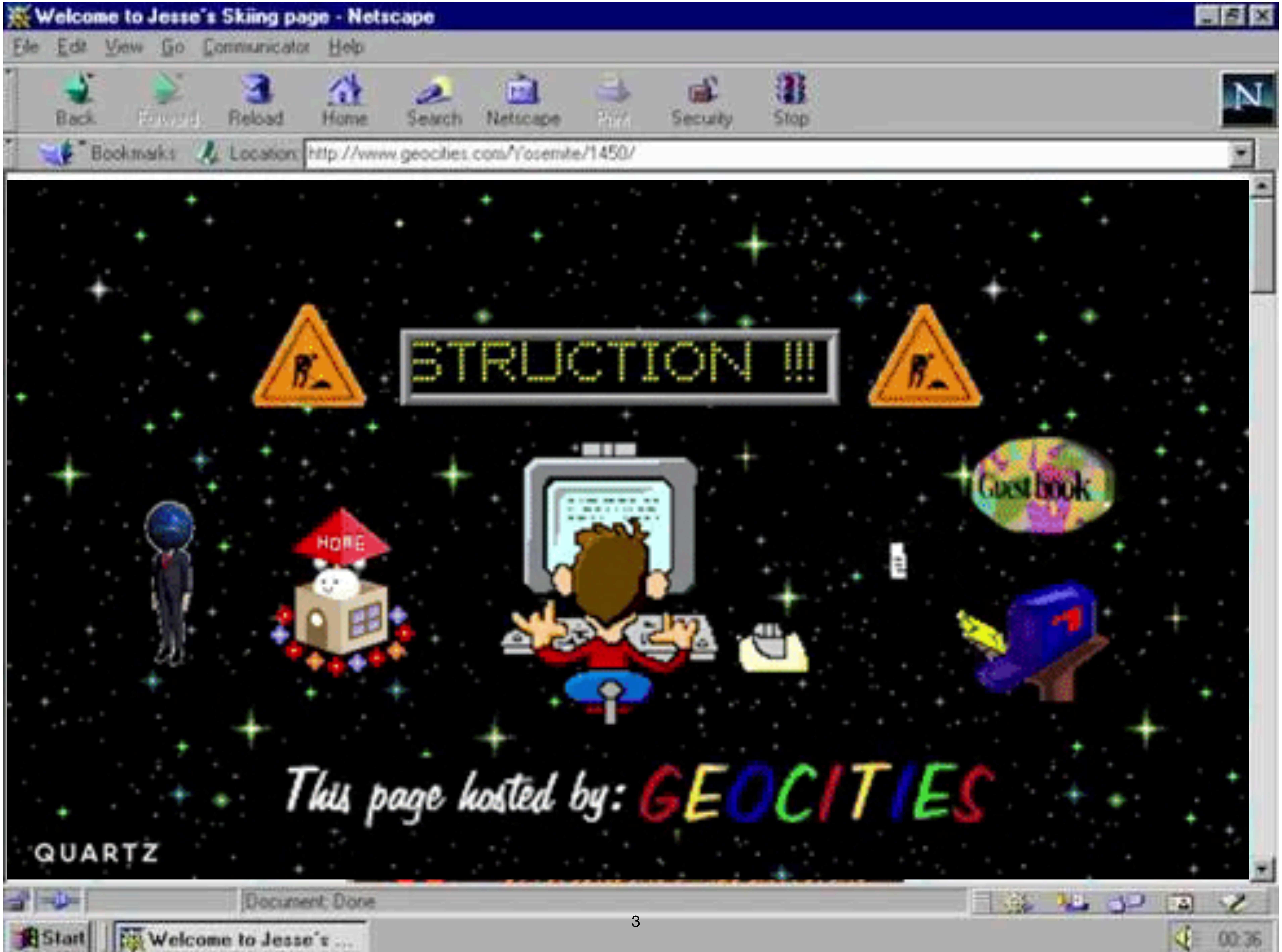
Qibin Liu[1], Chase Shimmin[2], Xiulong Liu[3], Eli Shlizerman[3], Shih-Chieh Hsu[3]

[1]SJTU, Shanghai [2]Yale University [3]UWashington

ML4Jets 2022 @ Rutgers **Wednesday, Nov 2**

Disclaimer:

Very Preliminary Results!!


Welcome to Jesse's Skiing page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.geocities.com/Yosemite/1450/
STRUCTION !!!
HOME
Guest book
This page hosted by: GEOCITIES
QUARTZ
Document: Done
Start
Welcome to Jesse's ...
00:36

VQ-VAE: Introduction

Why VQ-VAE?

- Began as conversations w/ our CS colleagues, Eli & Xiaolong :
- CaloFlow basically solved the calogan dataset
  - But it was slow….

    …until it wasn't!
- Eli &al suggested we look into VQ-VAE to handle scaling expensive generative models
- VQ-VAE in particular has been used on some complex generative tasks (e.g. Dall-E)

➡ Some earlier talks (Vinicius, Jesse) already discussed some of the advantages of a two-step model!!!

VQ-VAE Overview

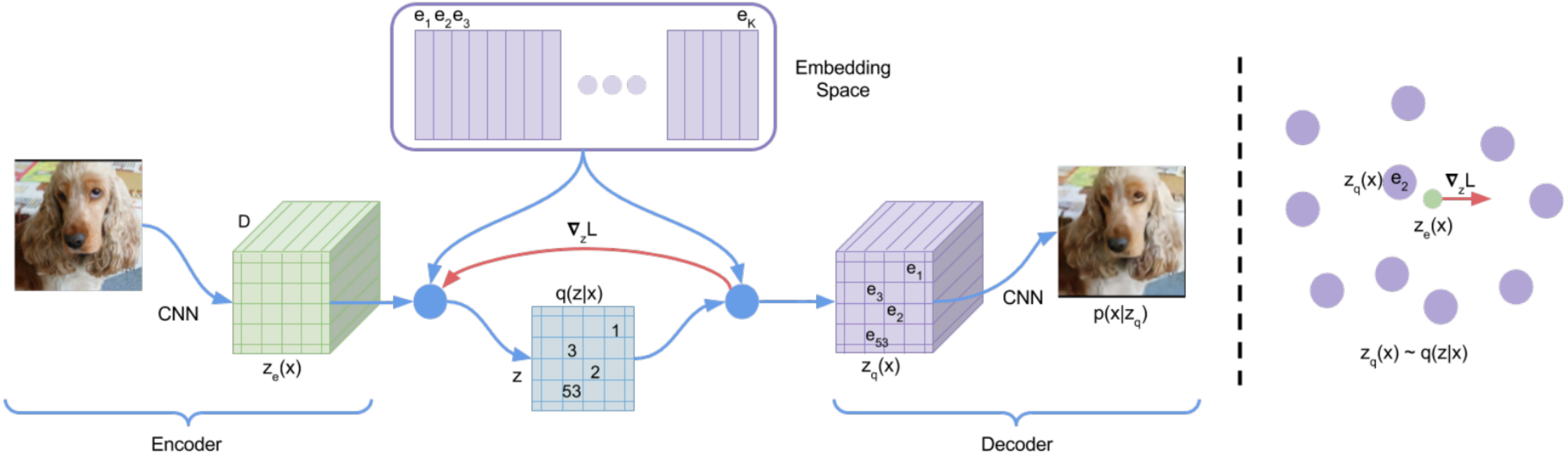
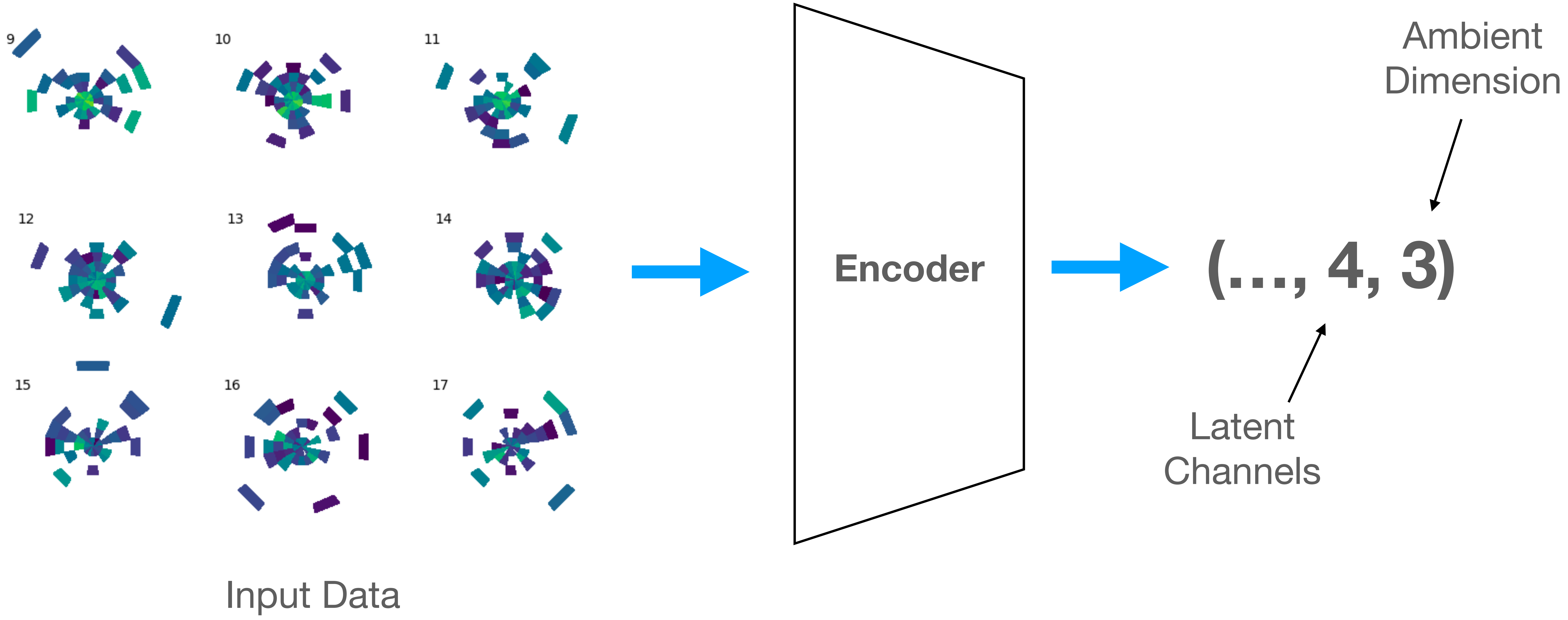


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

"Neural Discrete Representation Learning" Van den Oord et al. (2017)

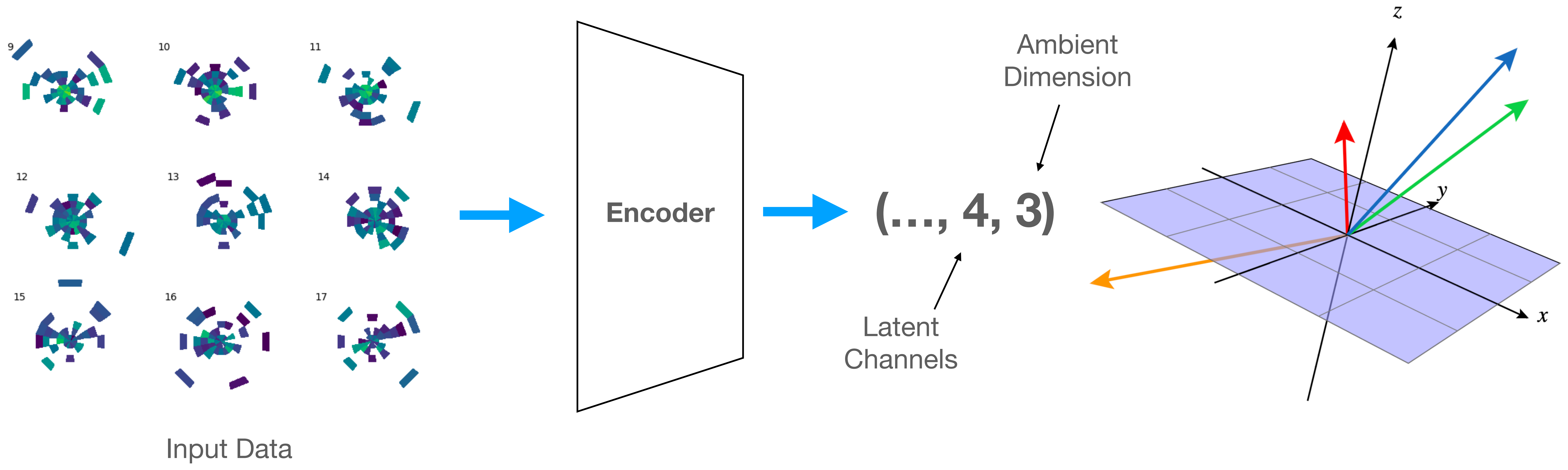
VQ-VAE Forward Pass

Step I: Regular encoder maps data to latent space



VQ-VAE Forward Pass

Step I: Regular encoder maps data to latent space



VQ-VAE Forward Pass

Codebook Size

$$\{e_i = (x_i, y_i, z_i) : i = 1..N\}$$

(Illus. w/ N=6)

Step II: Latent are compared to “code book” embeddings

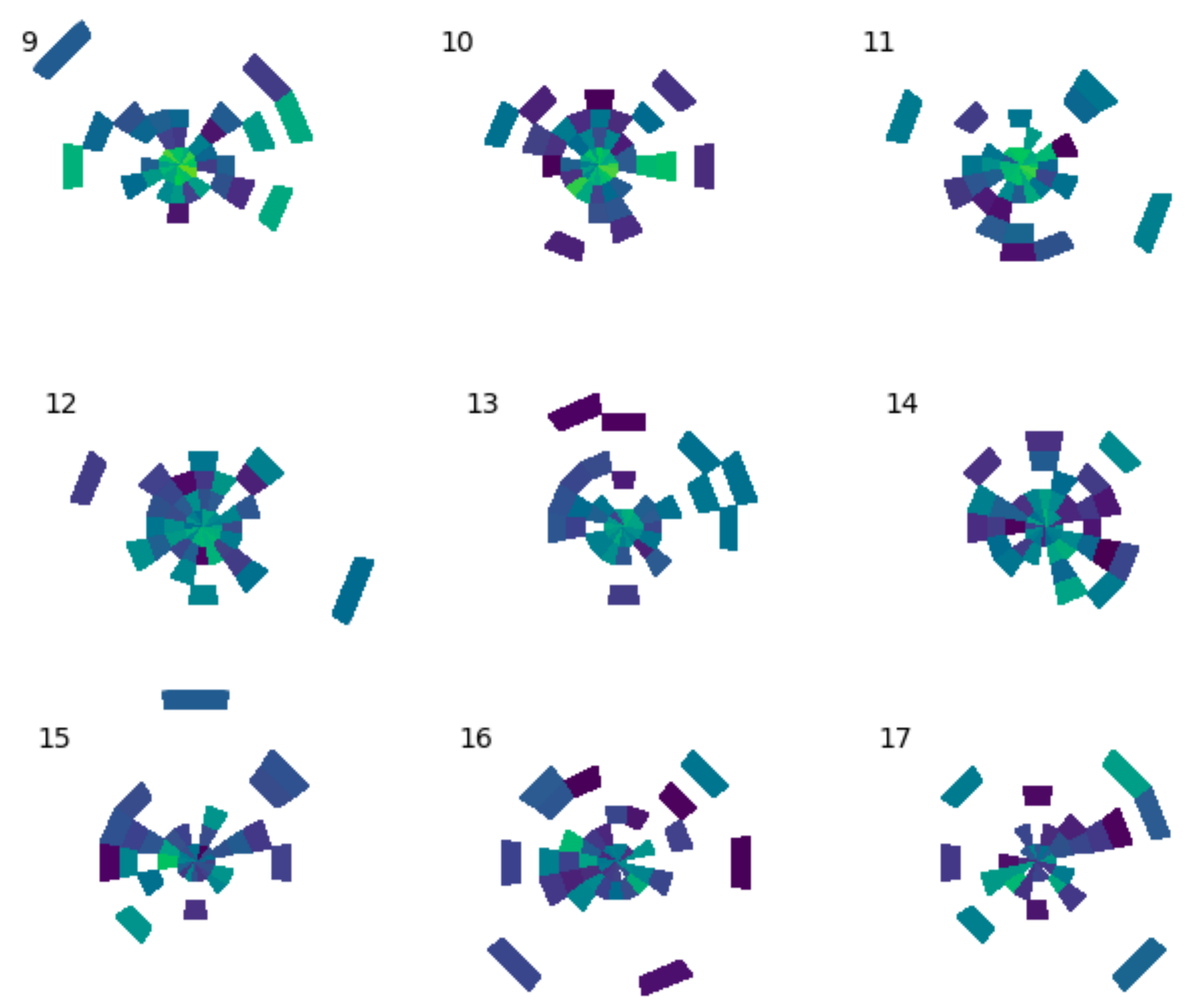
Codebook
Size

$$\{e_i = (x_i, y_i, z_i) : i = 1..N\}$$

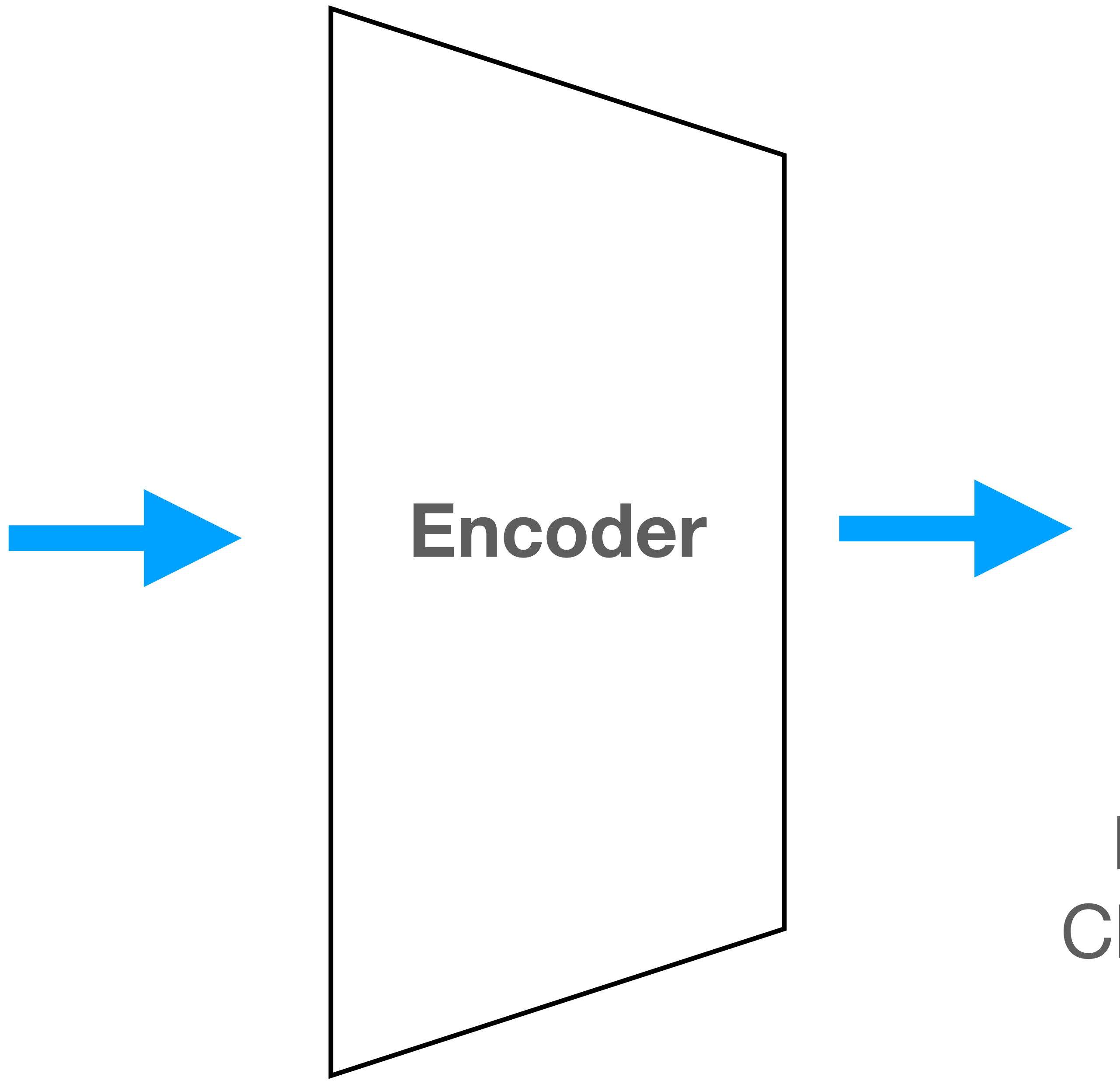
(Illus. w/ N=6)

VQ-VAE Forward Pass

Step II: Latent are compared to “code book” embeddings



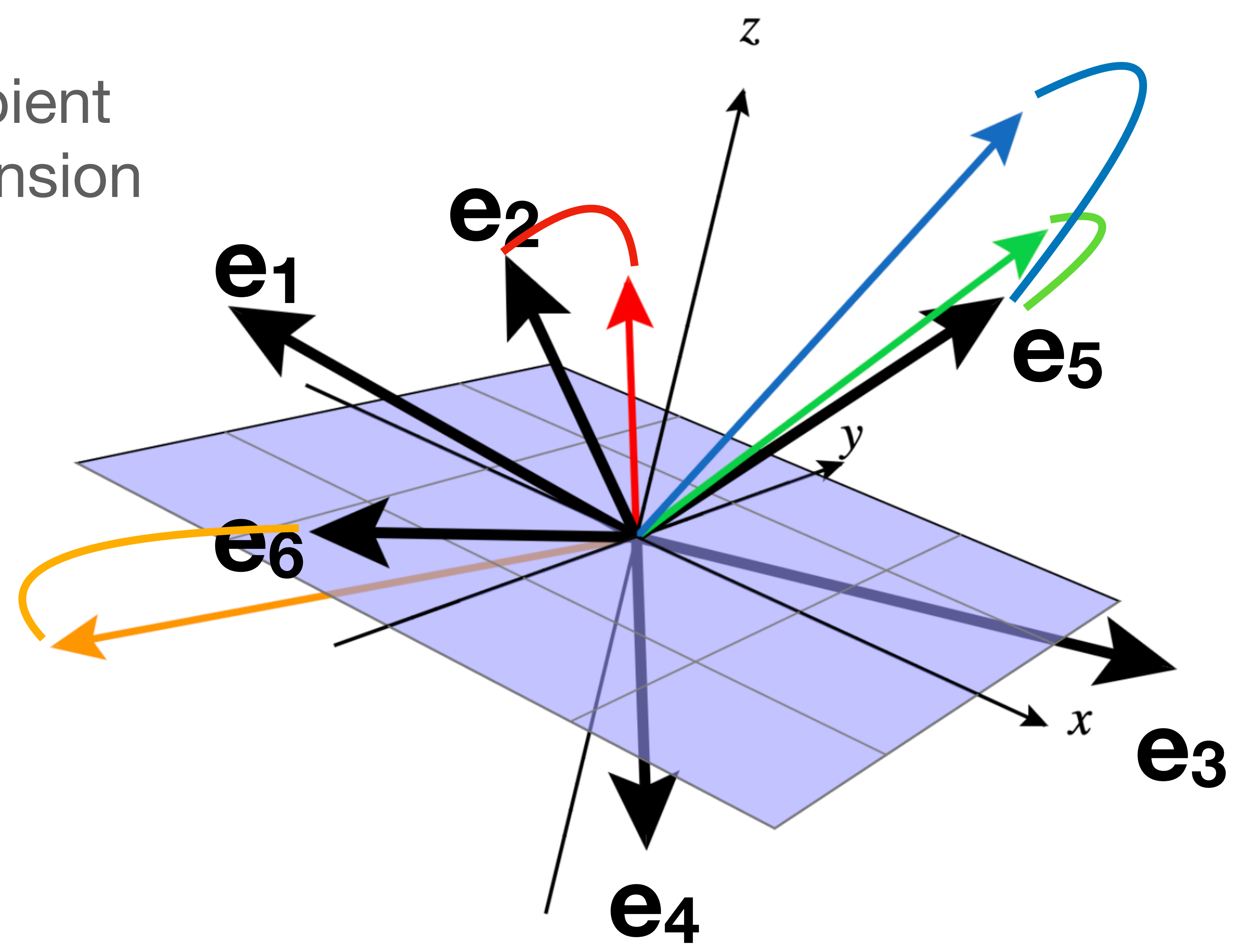
Input Data



Ambient
Dimension

(…, 4, 3)

Latent
Channels



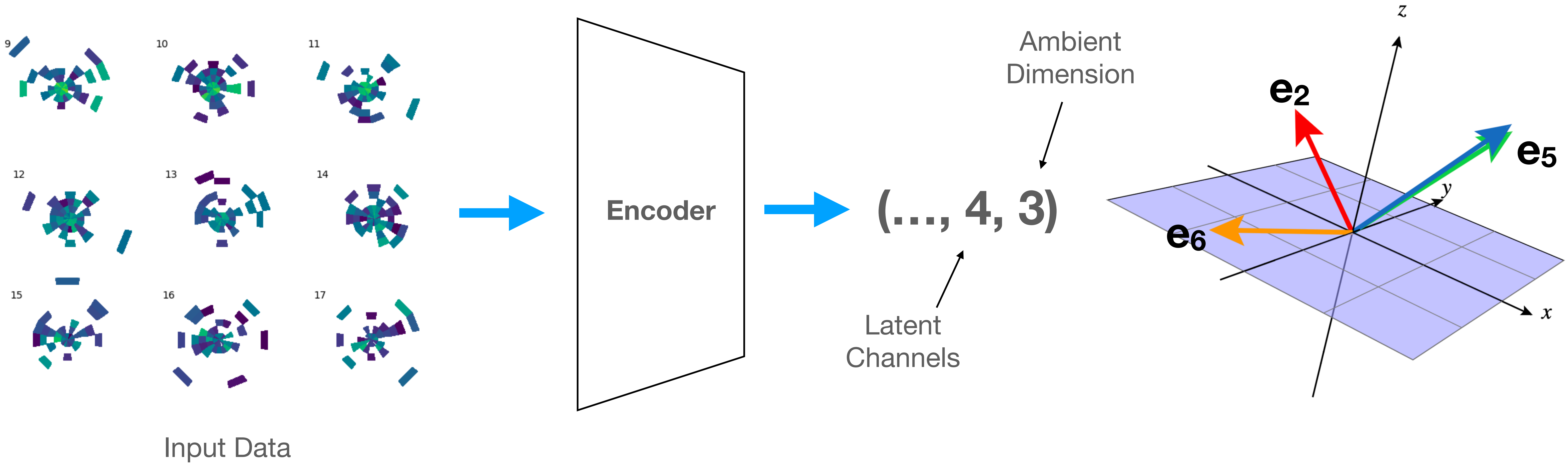
VQ-VAE Forward Pass

Codebook Size

$$\{e_i = (x_i, y_i, z_i) : i = 1..N\}$$

(Illus. w/ N=6)

Step III: Latent vectors are replaced with their nearest embedding



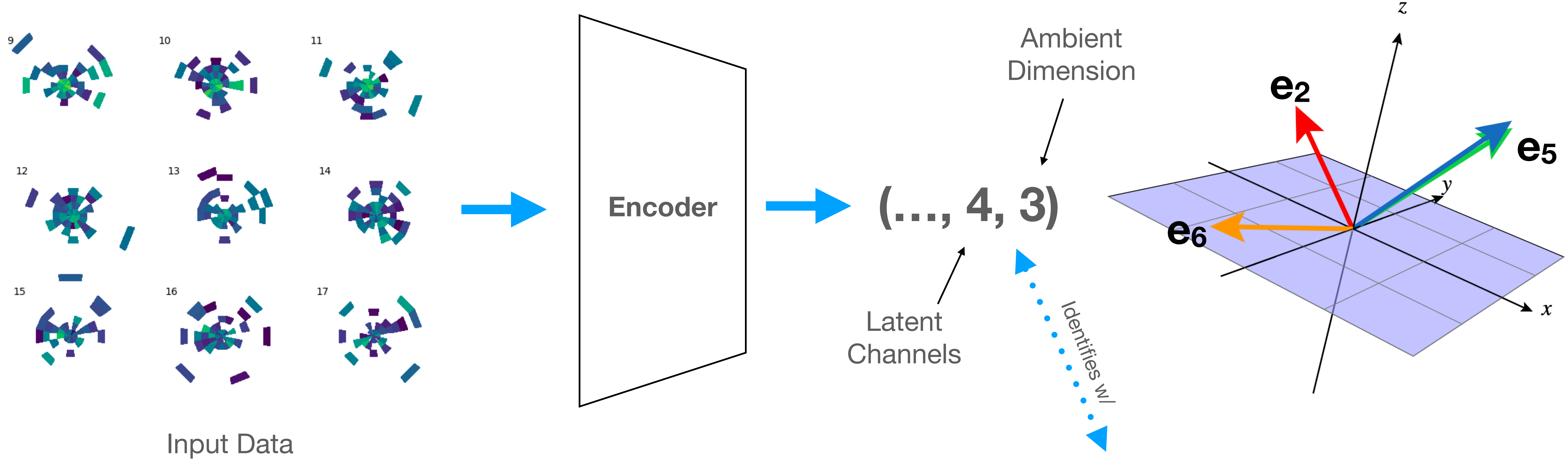
VQ-VAE Forward Pass

$$\{e_i = (x_i, y_i, z_i) : i = 1..N\}$$

Codebook Size

(Illus. w/ N=6)

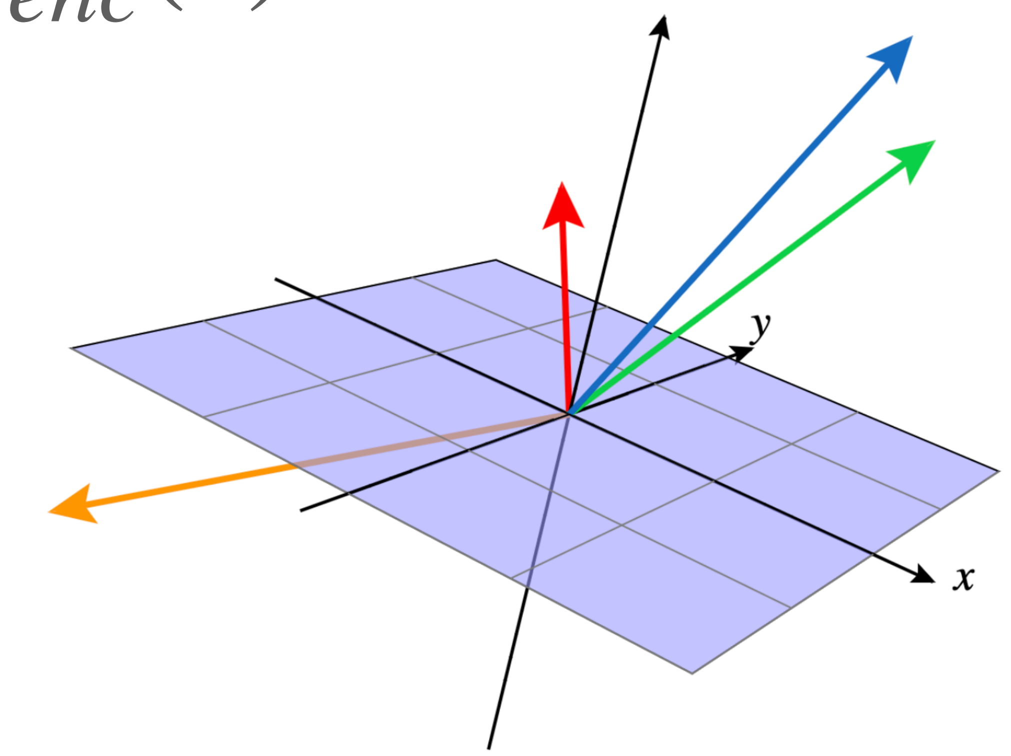
Step III: Latent vectors are replaced with their nearest embedding



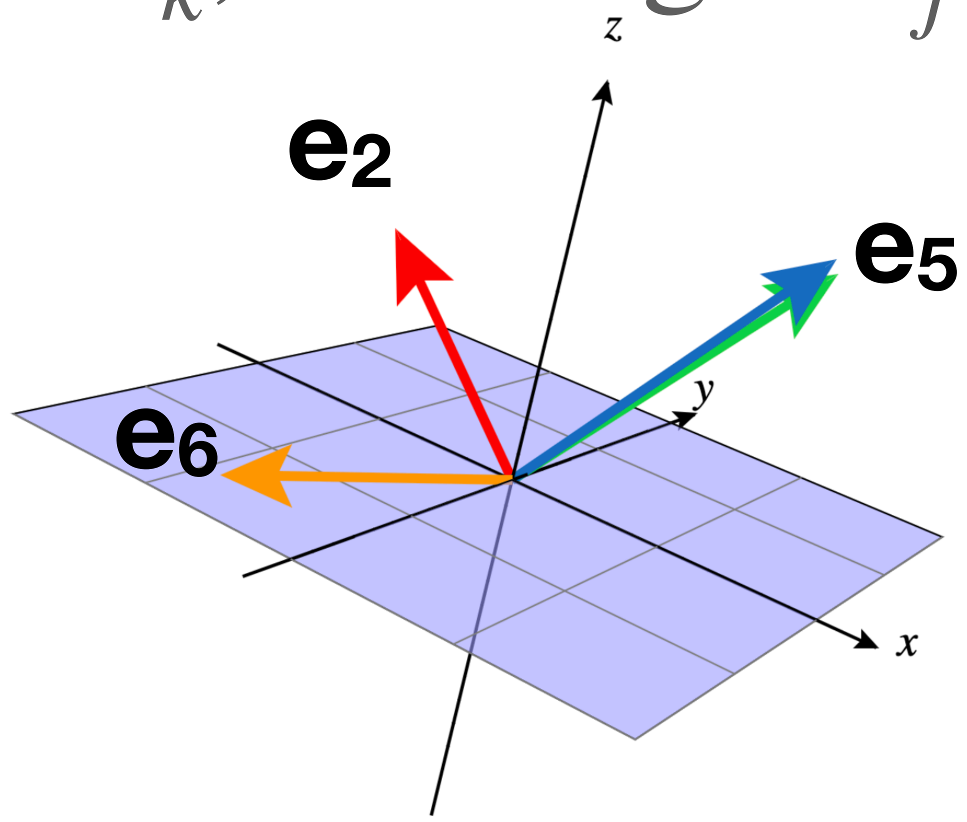
Quantized latent space: **(…, 4) : (2,6,5,5)** $\in Z, \quad |Z| = 4^N$

VQ-VAE Objective

$z_{enc}(x)$: raw encoder output



$$z_q(x) = e_k; \;\; k = \operatorname{argmin}_j | z_e(x) - e_j |$$



$$L = \log p\left(x \,|\, z_q(x)\right) + \left|\mathrm{sg}[z_{enc}(x)] - e\right|^2 + \beta \left|z_{enc}(x) - \mathrm{sg}[e]\right|^2$$

Reconstruction Loss

(e.g. MSE, GAN, perceptual loss, etc)

Vector Quantization

Dictionary learning
//
Update codebook

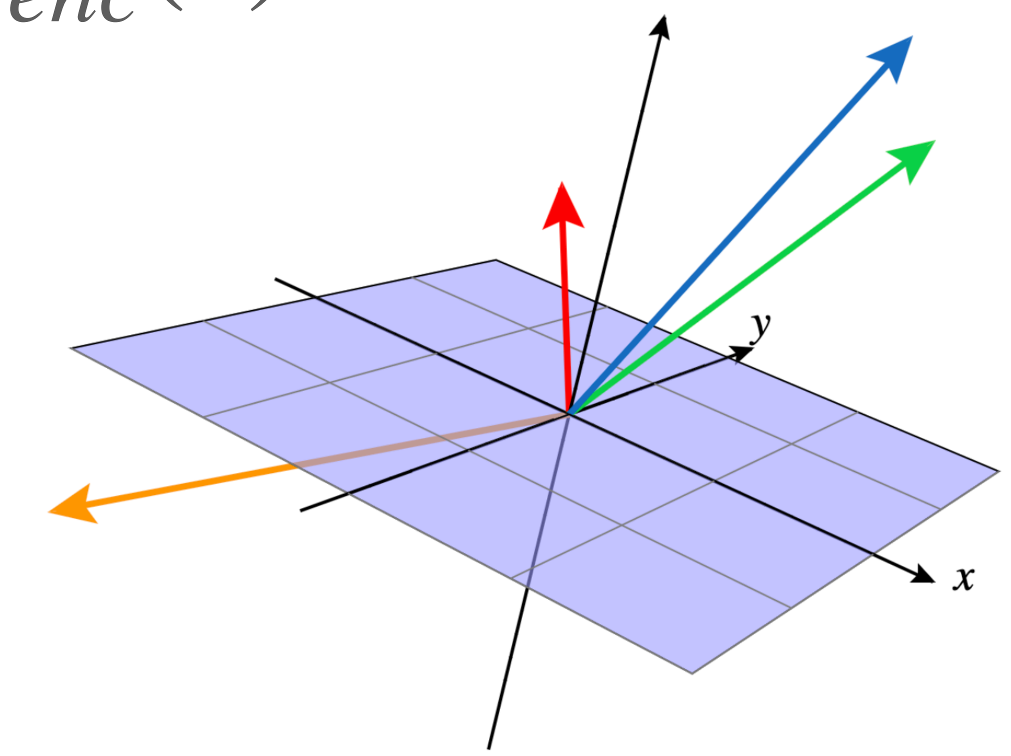
Commitment Loss

Tries to keep encoder predictions close to codebook values.

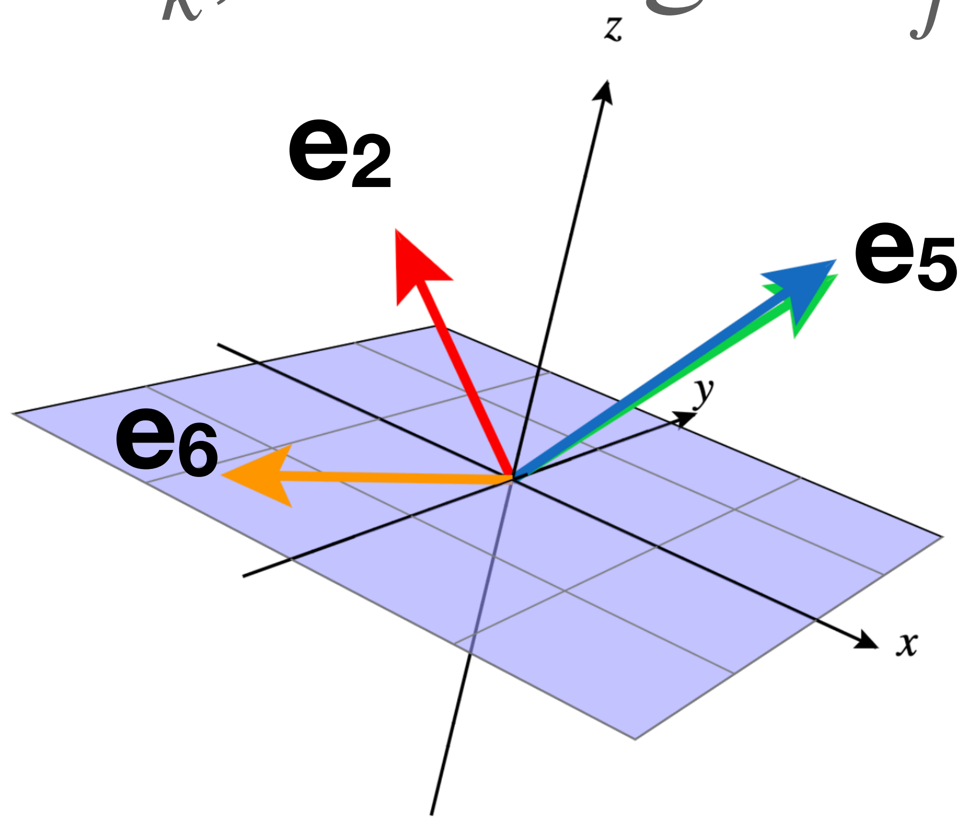
* *sg[] = stop gradient*

VQ-VAE Objective

$z_{enc}(x)$: raw encoder output



$z_q(x) = e_k;\ \ k = \mathrm{argmin}_j \,|\, z_e(x) - e_j \,|$



$$L = \log p\left(x \,|\, z_q(x)\right) + \left| \mathrm{sg}[z_{enc}(x)] - e \right|^2 + \beta \left| z_{enc}(x) - \mathrm{sg}[e] \right|^2 + C$$

Reconstruction Loss

(e.g. MSE, GAN, perceptual loss, etc)

Vector Quantization

Dictionary learning
//
Update codebook

Commitment Loss

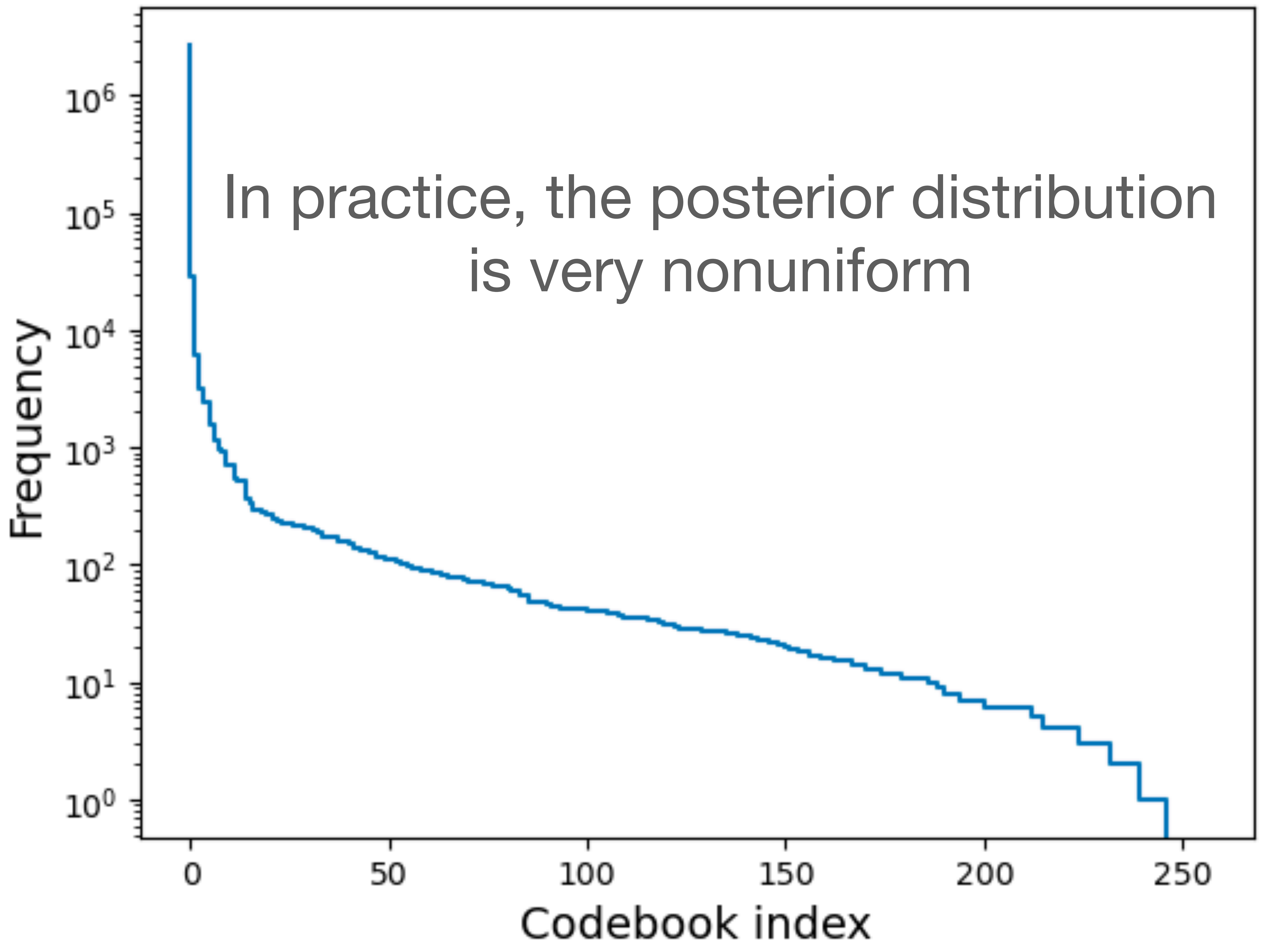
Tries to keep encoder predictions close to codebook values.

“KL” Term

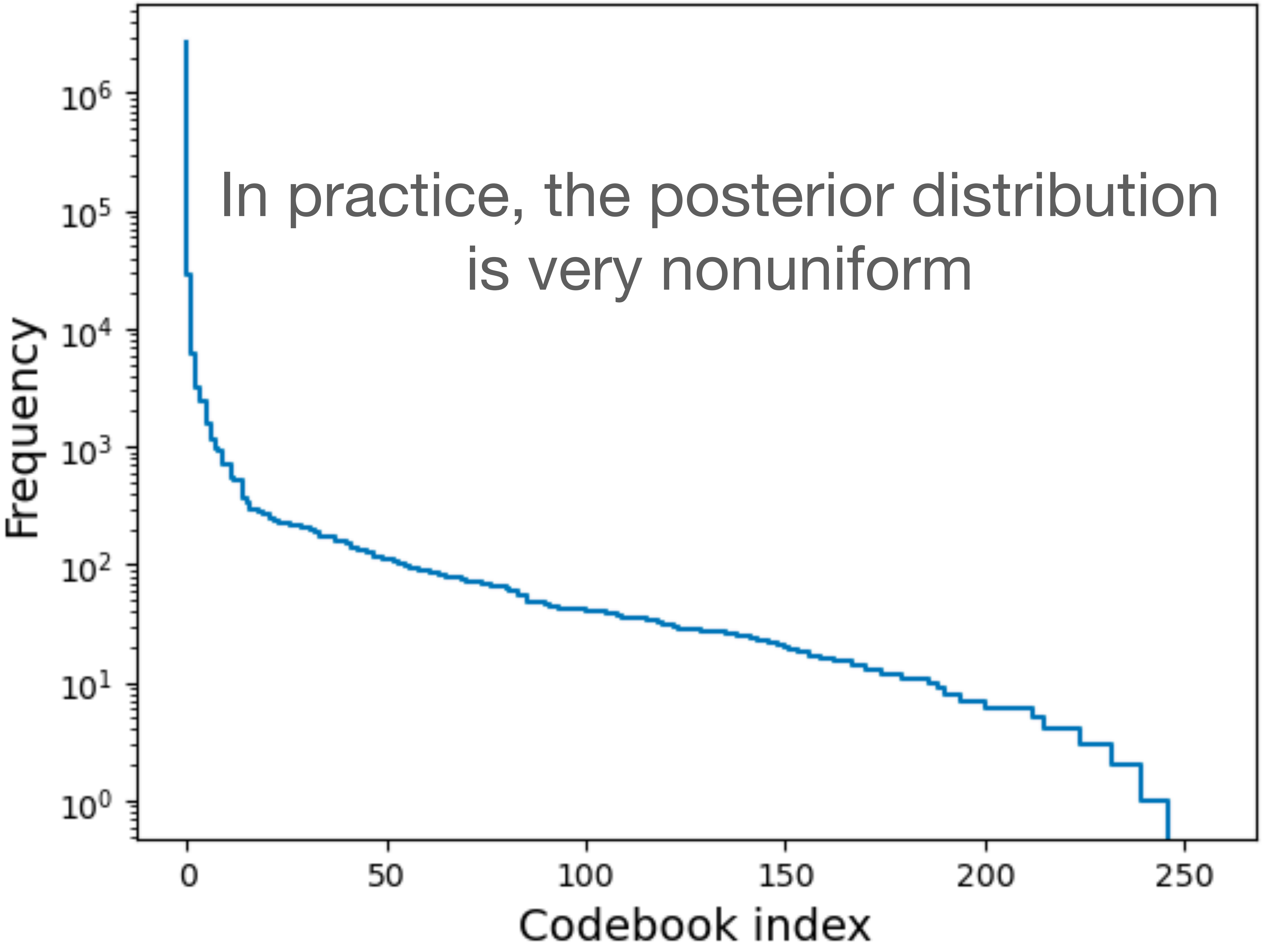
Assuming uniform prior!

** sg[] = stop gradient*

VQ-VAE Latent Distribution



VQ-VAE Latent Distribution



But that's okay!

- Once VQVAE achieves good reconstruction, a separate generative model is trained to learn a new (discrete) prior distribution.
- This means the **representation learning is factorized** from the generative model.
  - Can train one VQVAE, and experiment with many different generative priors
  - Obviates the problem of tuning "beta-VAE"

CaloChallenge Dataset 1

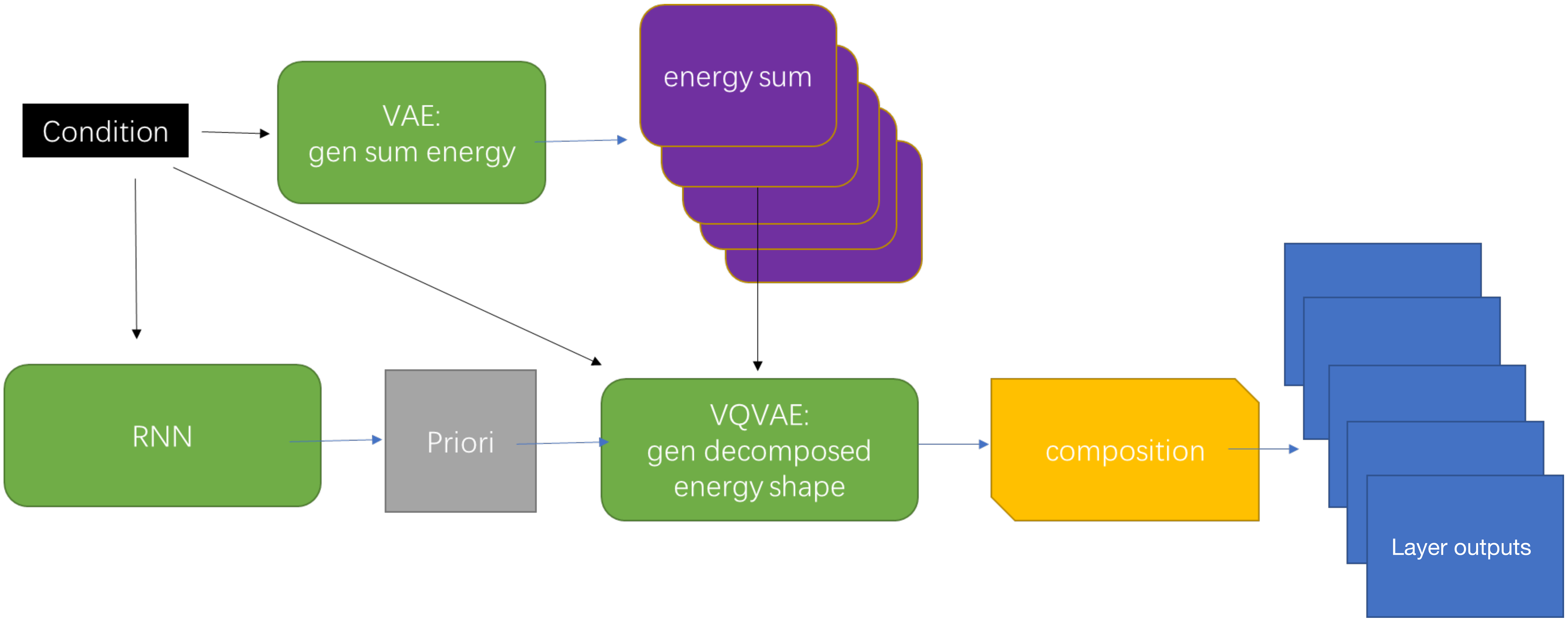
DS 1: Strategy

Our strategy:

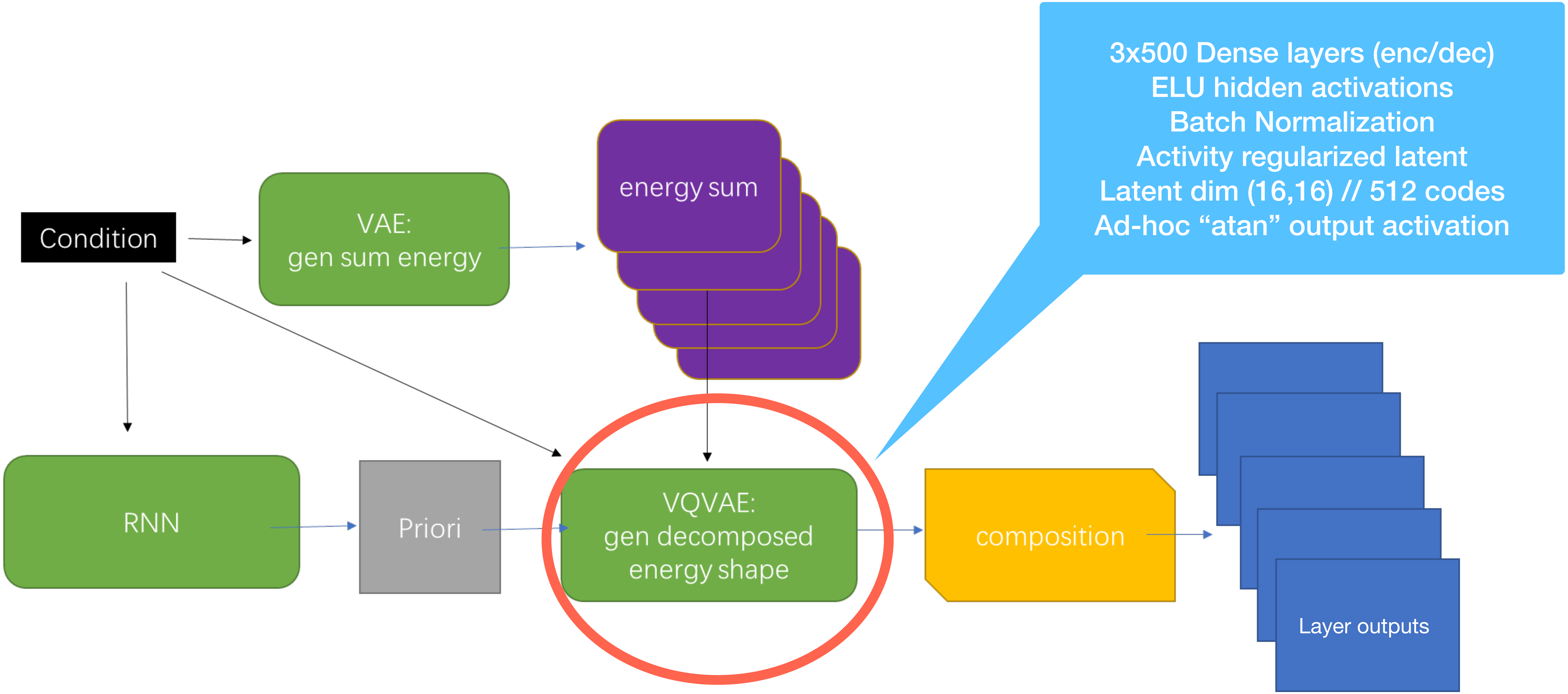
- Fairly low dimensional: at 368-533 voxels, **no regular structure**
- **Fully-connected encoder/decoder** architecture
- Normalize each layer in each event (h/t caloFlow)
- Predict layer-wise energy and voxel "shape" information separately
  - The 5 layer energies are predicted using a standard cVAE
  - The fractionalized voxel energies predicted using VQVAE

- VQVAE encodes to **discrete 16 dimensional latent** (~96% compression)
  - +4 dim for layer energy VAE
- VQ prior modeled with **autoregressive conditional RNN**

DS1: Architecture



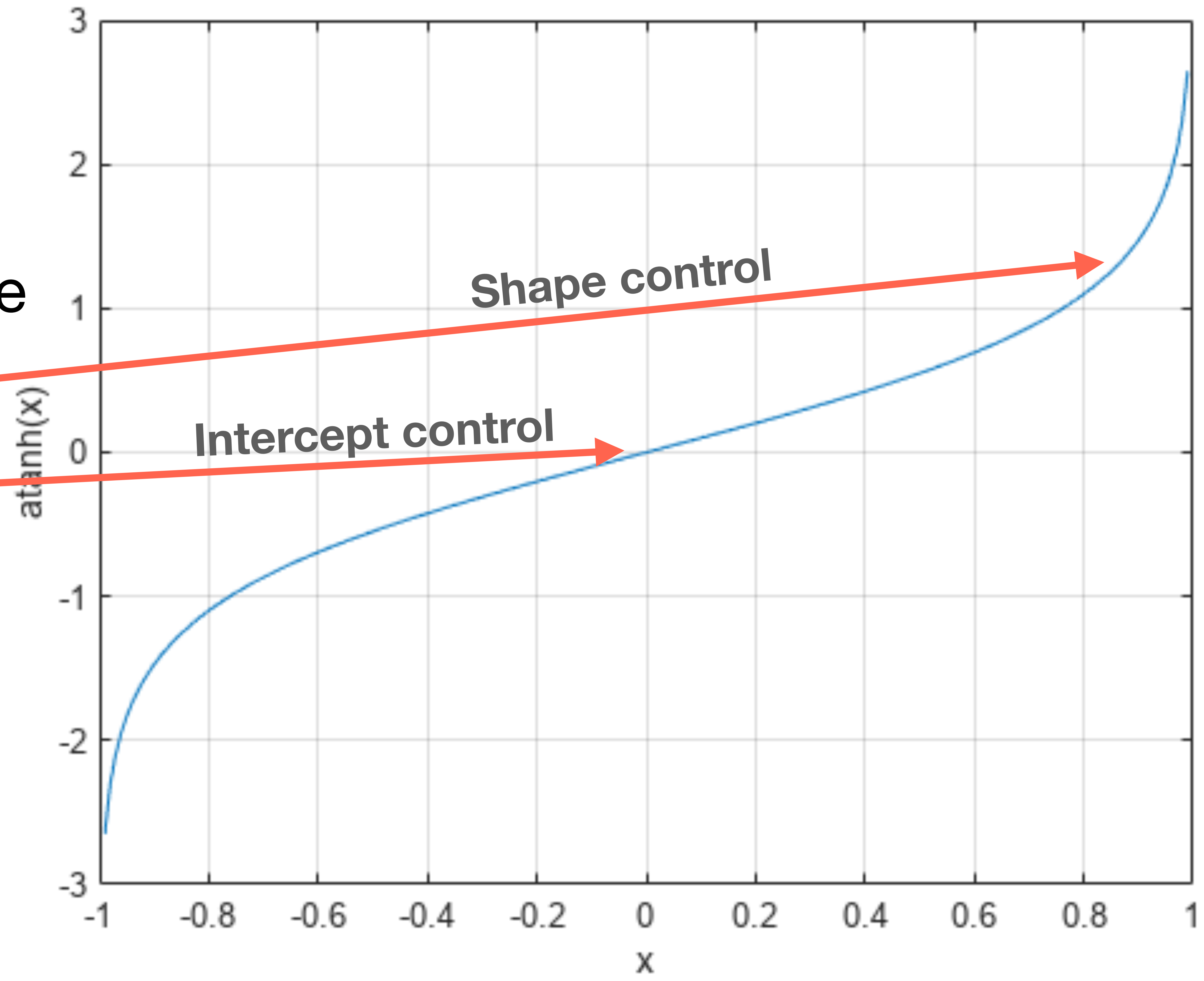
DS1: Architecture



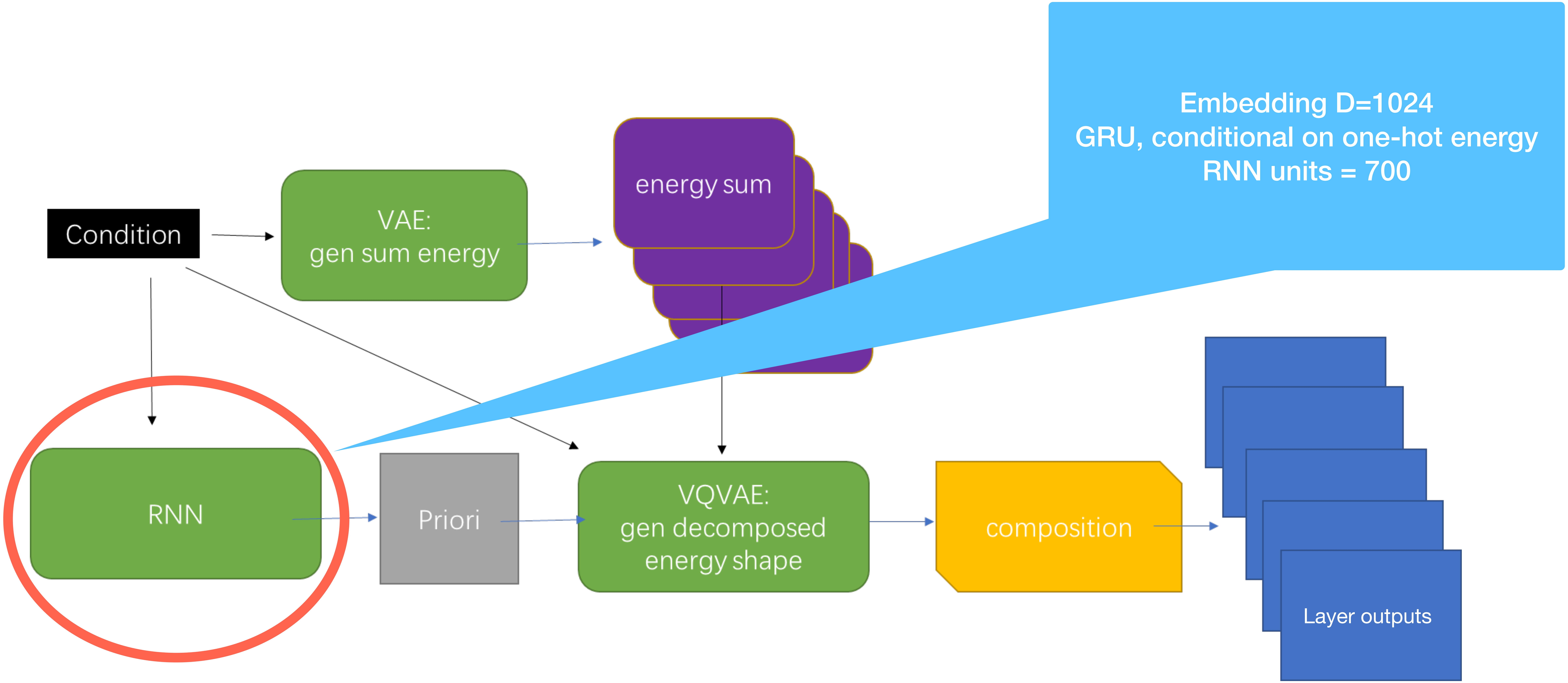
Arctangent Activation

- Useful output activation
- Increases sensitivity at high energy
- Attractor to zero
- Critical to model both high dynamic range and sparsity

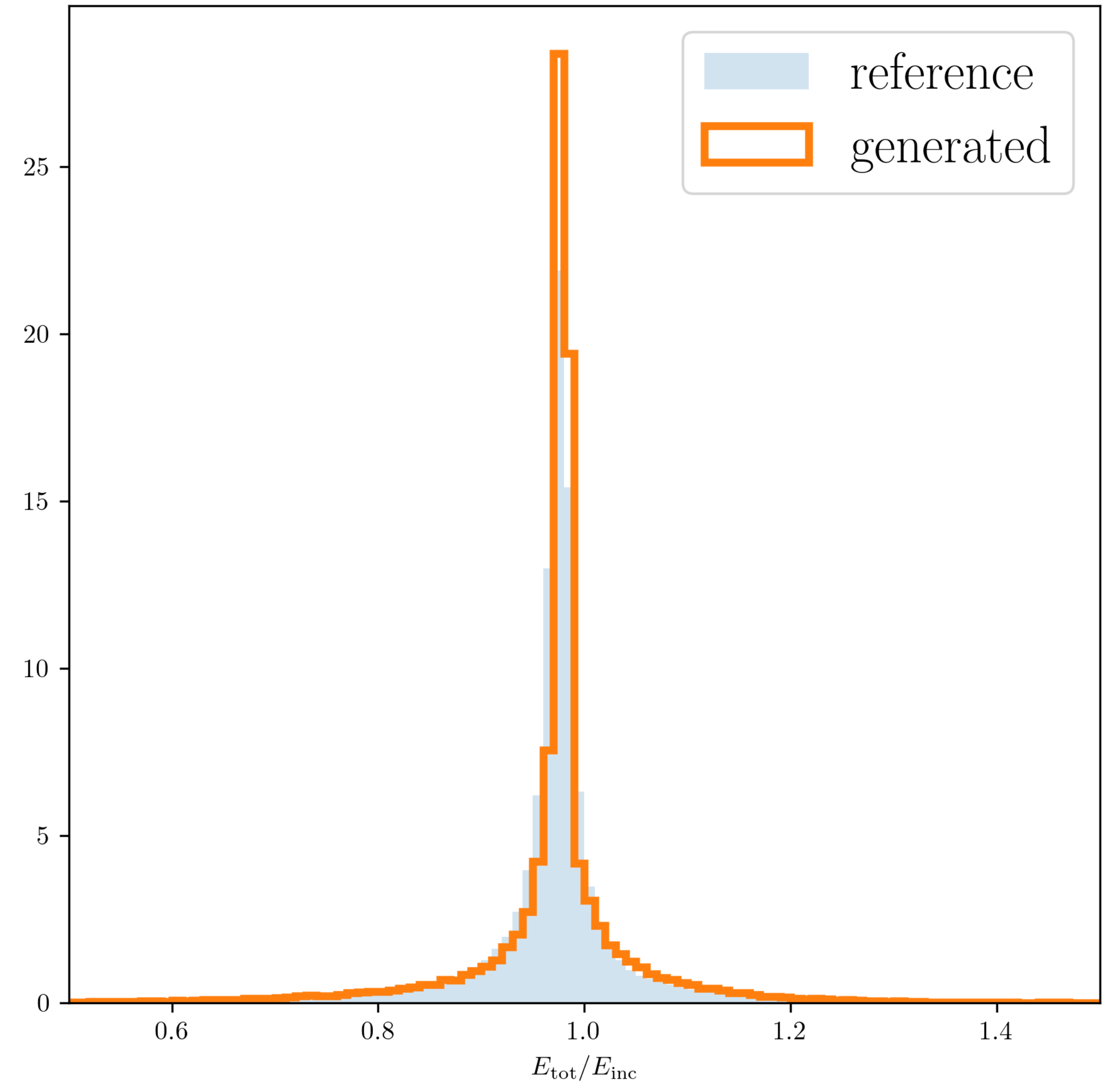
```
S=1E-4
T=0.2
K=2-S
B=T*S-1
A=1/(np.arctanh(K+B)-np.arctanh(B))
C=-A*np.arctanh(B)
def TRANS(x):
    return A*np.arctanh(K*x+B)+C
def ITRANS(y):
    return (np.tanh((y-C)/A)-B)/K
```

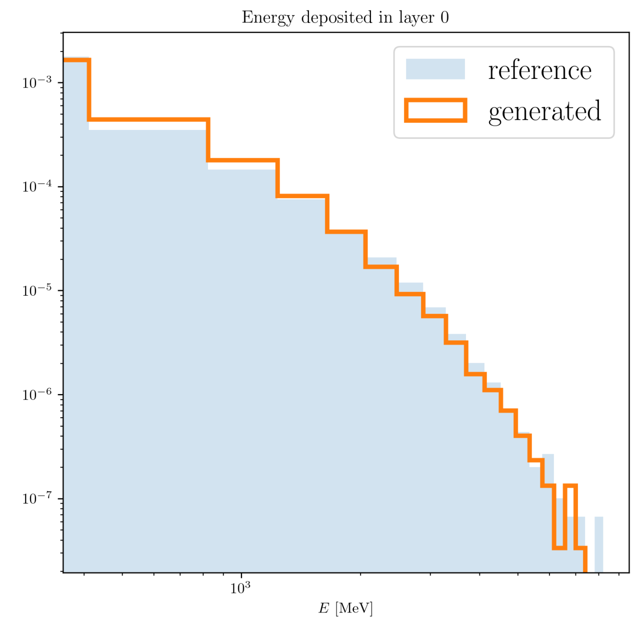


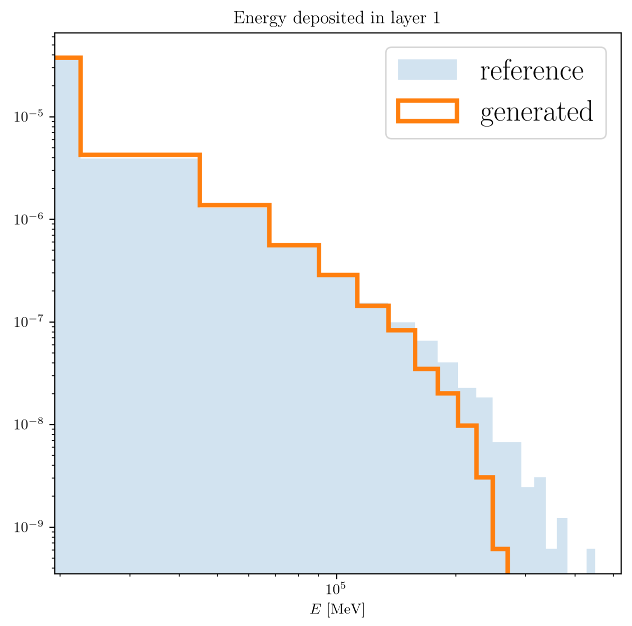
DS1: Architecture

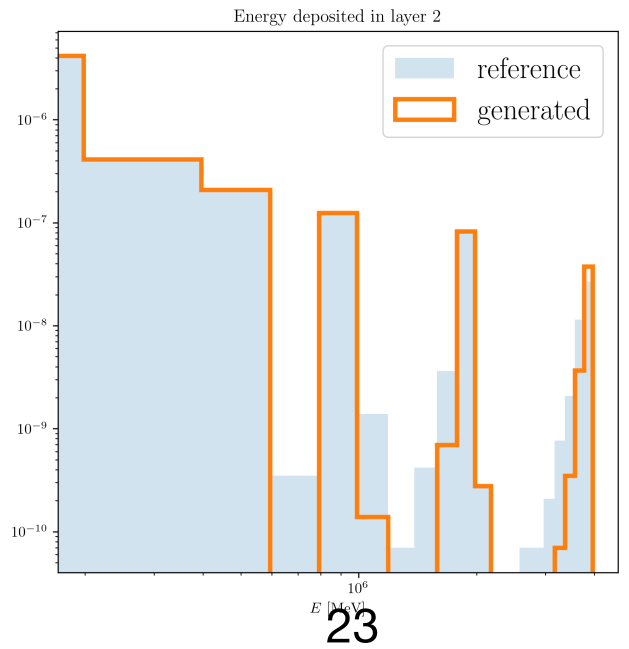


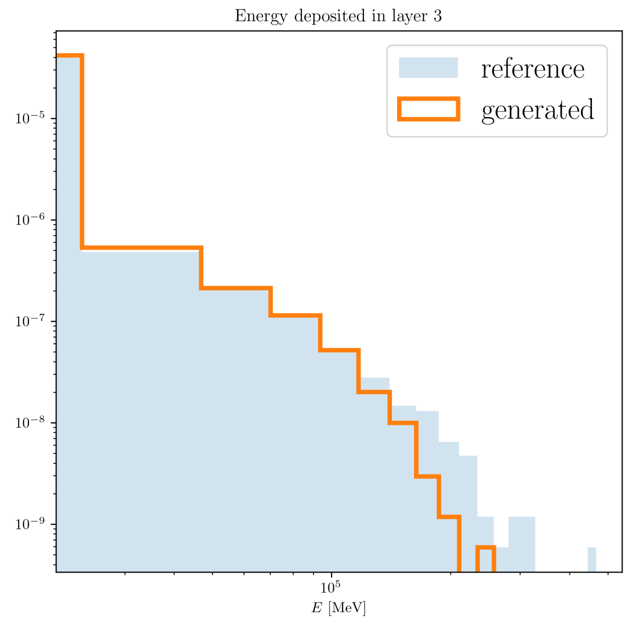
Dataset 1: Energy metrics

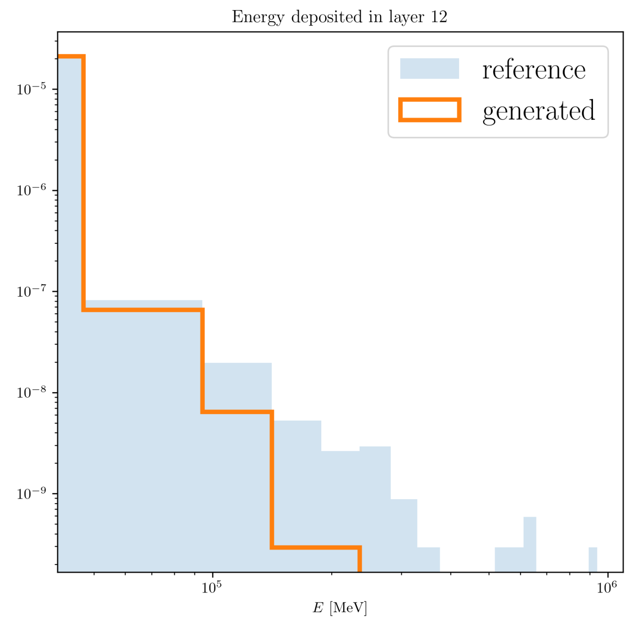




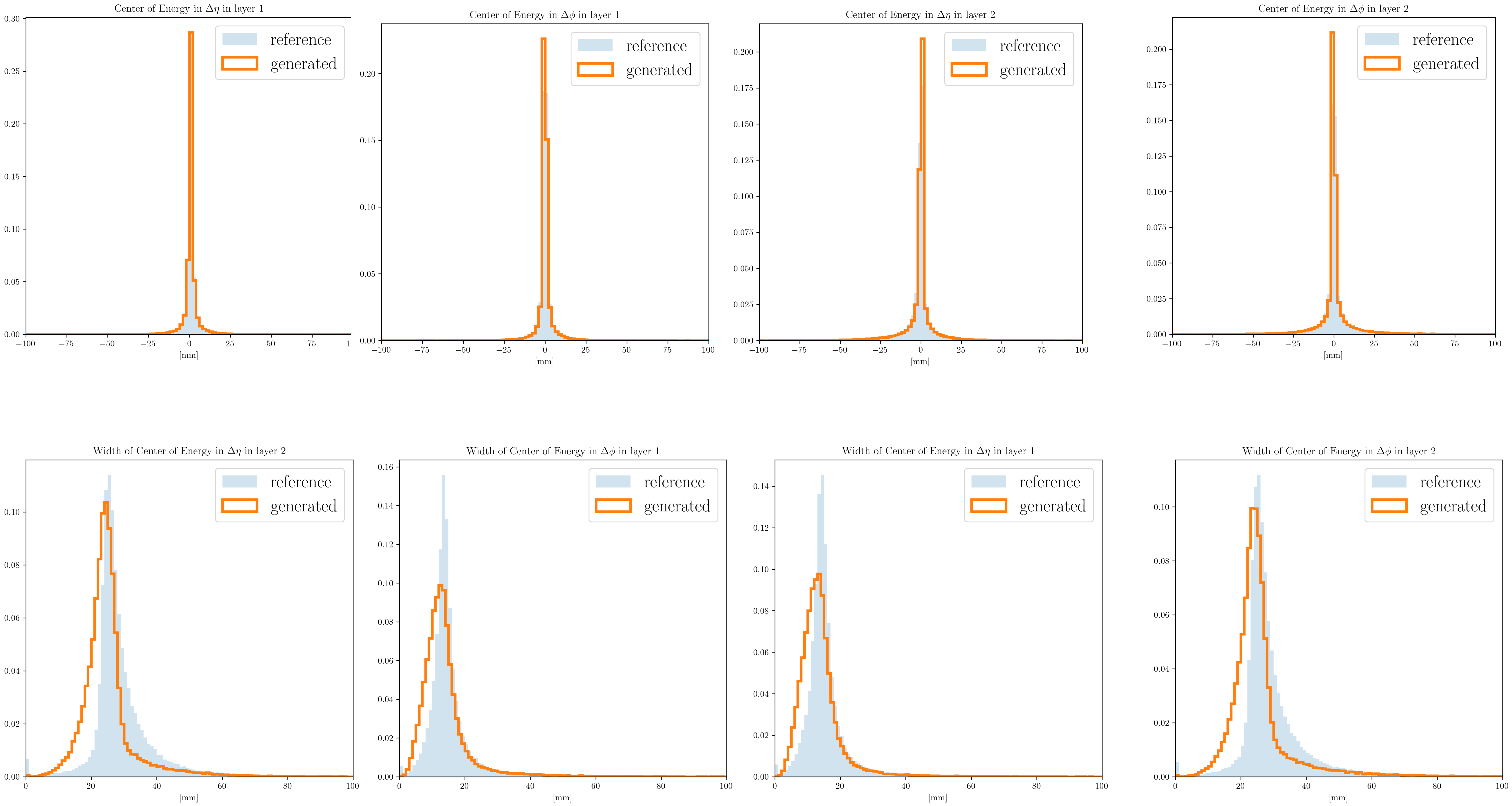




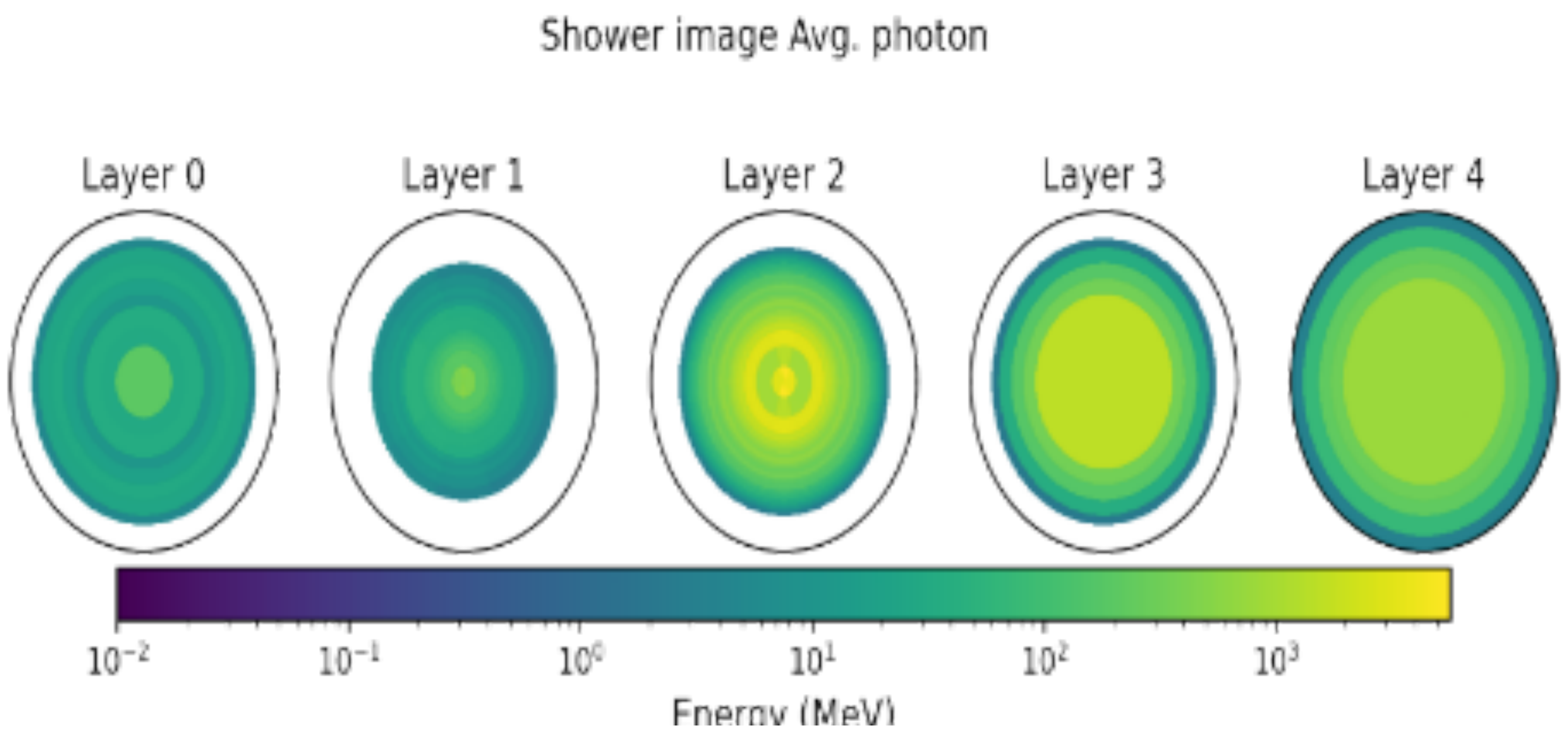


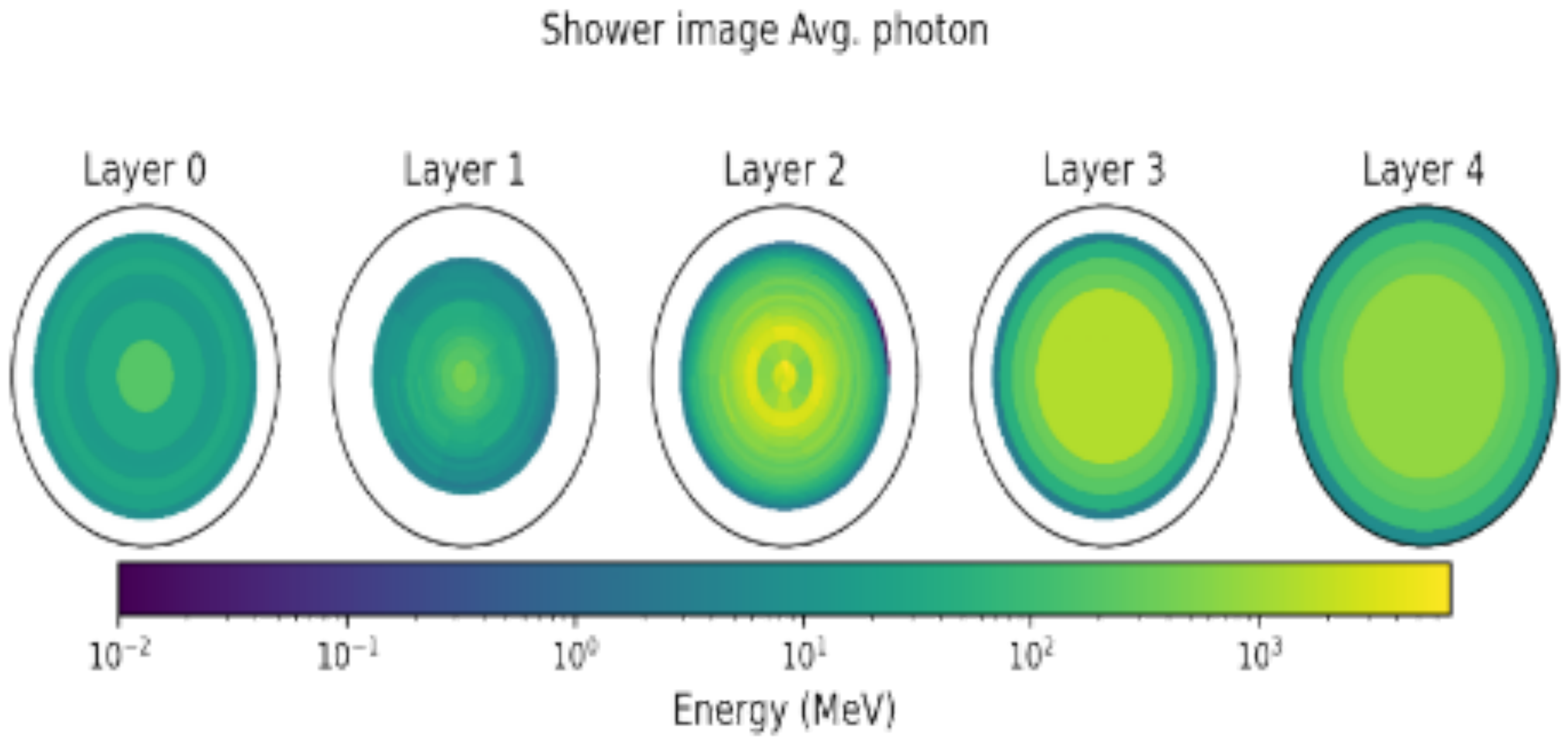


Dataset 1: Shape metrics

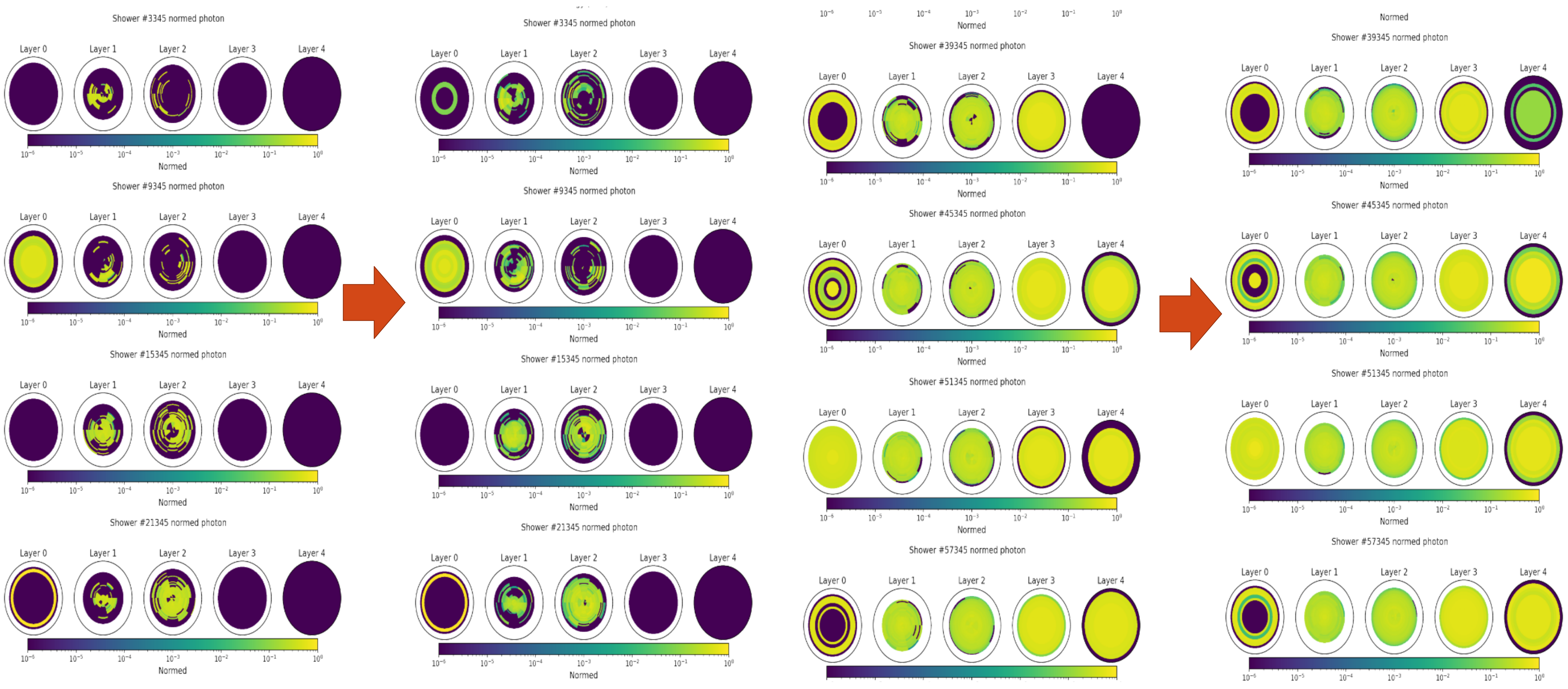


DS1: Reconstruction Performance





DS1: Reconstruction Performance



Dataset 1: Generation Time

- on A100 GPU: 0.04725 ms/evt
  - S-VQVAE: 0.0107 ms/evt +
  - E-VQE: 0.00025 ms/evt +
  - P-RNN: 0.0363 ms/evt

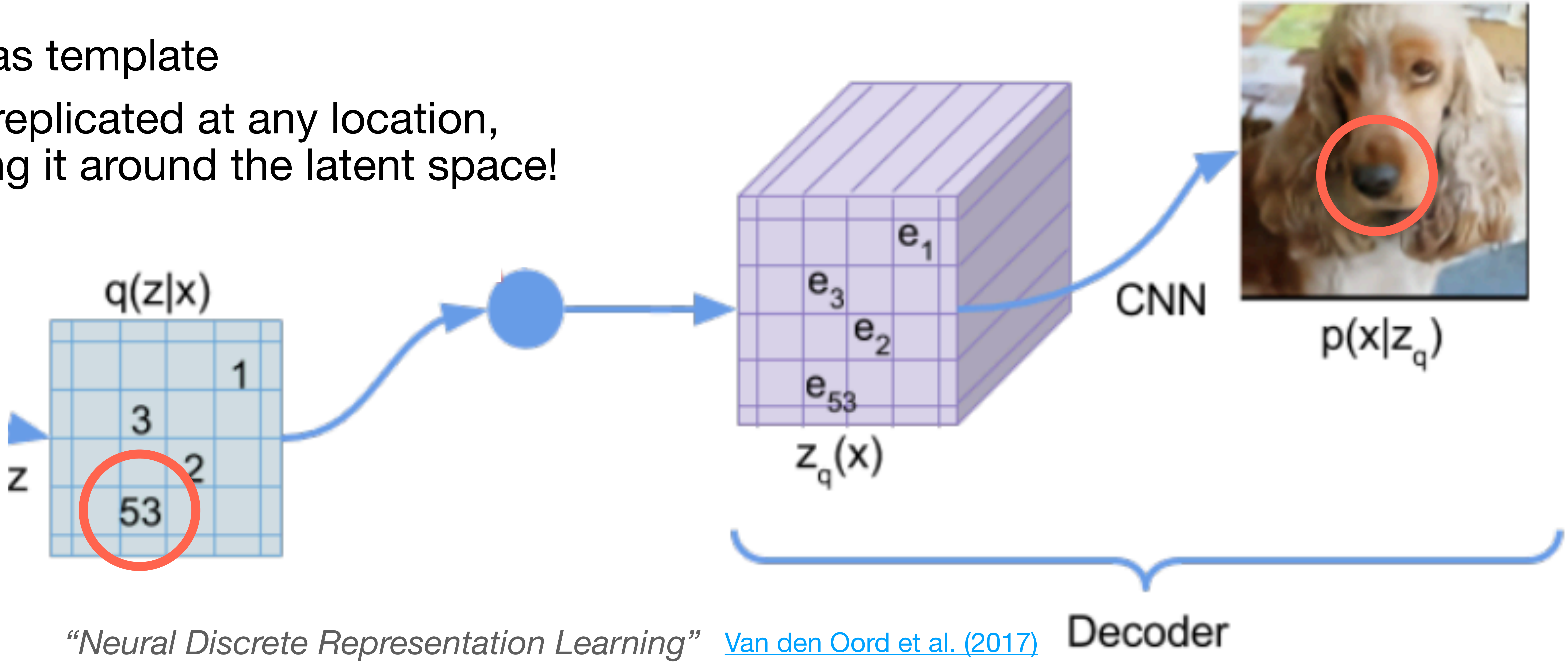
Prior model sampling is the bottleneck, by far

(Aside) VQ-VAE + Equivariance

"Neural Discrete Representation Learning" [Van den Oord et al. (2017)]

(Aside) VQ-VAE + Equivariance

- VQVAE makes intuitive sense in the context of equivariant architectures
- E.g., perhaps the model has learned that **code 53** means "**dog nose**"
  - ➡ This code acts as template
  - ➡ Can be reused/replicated at any location, simply by moving it around the latent space!



"Neural Discrete Representation Learning" [Van den Oord et al. (2017)](#)

CaloChallenge: Dataset 2

CaloChallenge DS2+3: Strategy / Plans

- DS2 (and DS3) are **much larger datasets**:
  6.5k and 40.5k voxels, resp.
- Both have **regular cylindrical structure**
- Use **cylindrical convolution** to reduce spatial dimension
- VQVAE to reduce “information” dimension
- Generative model for prior: **TBD…**
  - Training previous autoregressive RNN with hundreds or thousands of steps is infeasibly slow.
  - We are investigating alternative models.

- Initial study on reconstruction only:
  - Fully-convolutional encoder/decoder
  - It’s possible to reduce the spatial dimension as low as ~300, but performance is not great
  - Can achieve good reconstruction fidelity with ~30% compression
  - We hope to combine with other generative models: score, transformer, maybe flow(?), for dimensional speedup

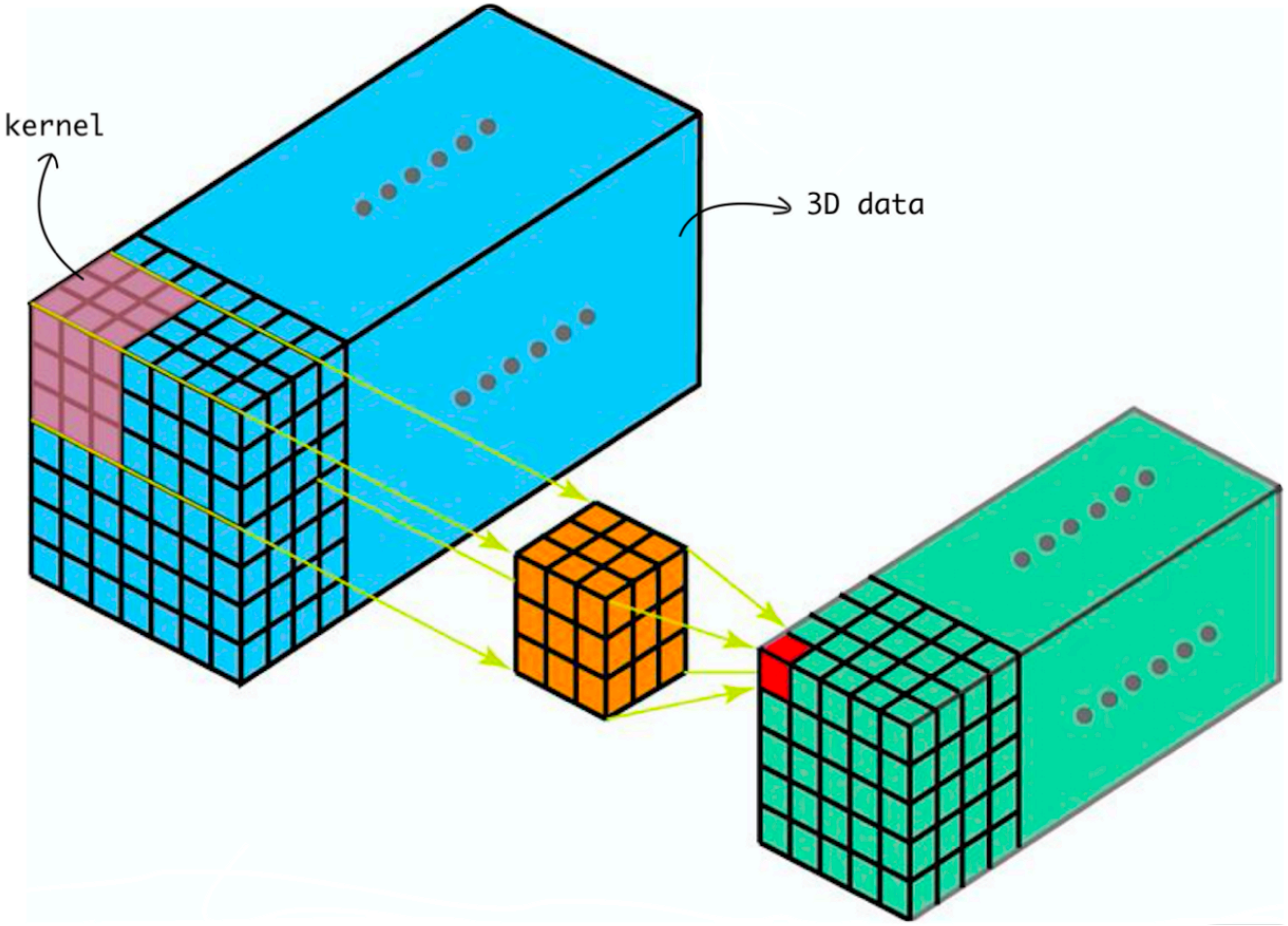
How to do Cylindrical Convolution?

Usual 3D convolution:

Input data is a (cubic) grid
kernel is a (cubic) grid

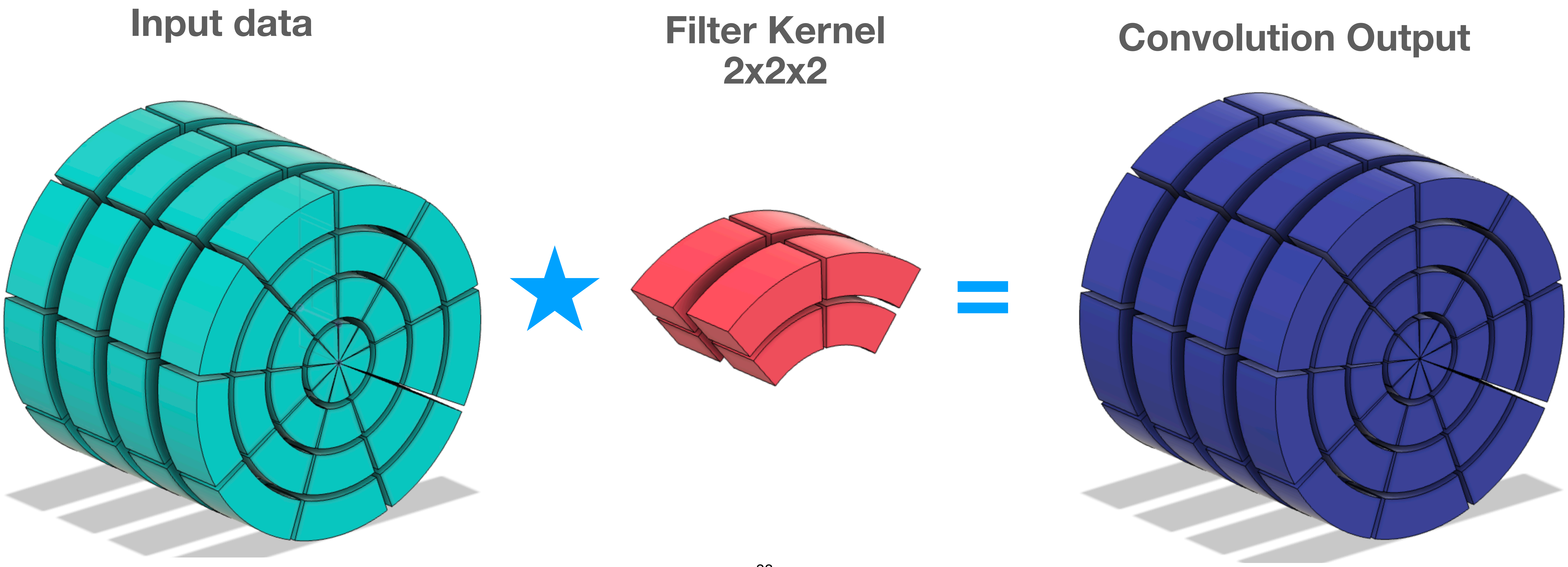
↓

Get a cubic grid back



How to do Cylindrical Convolution?

We would like to have a cylindrical grid as input/output!

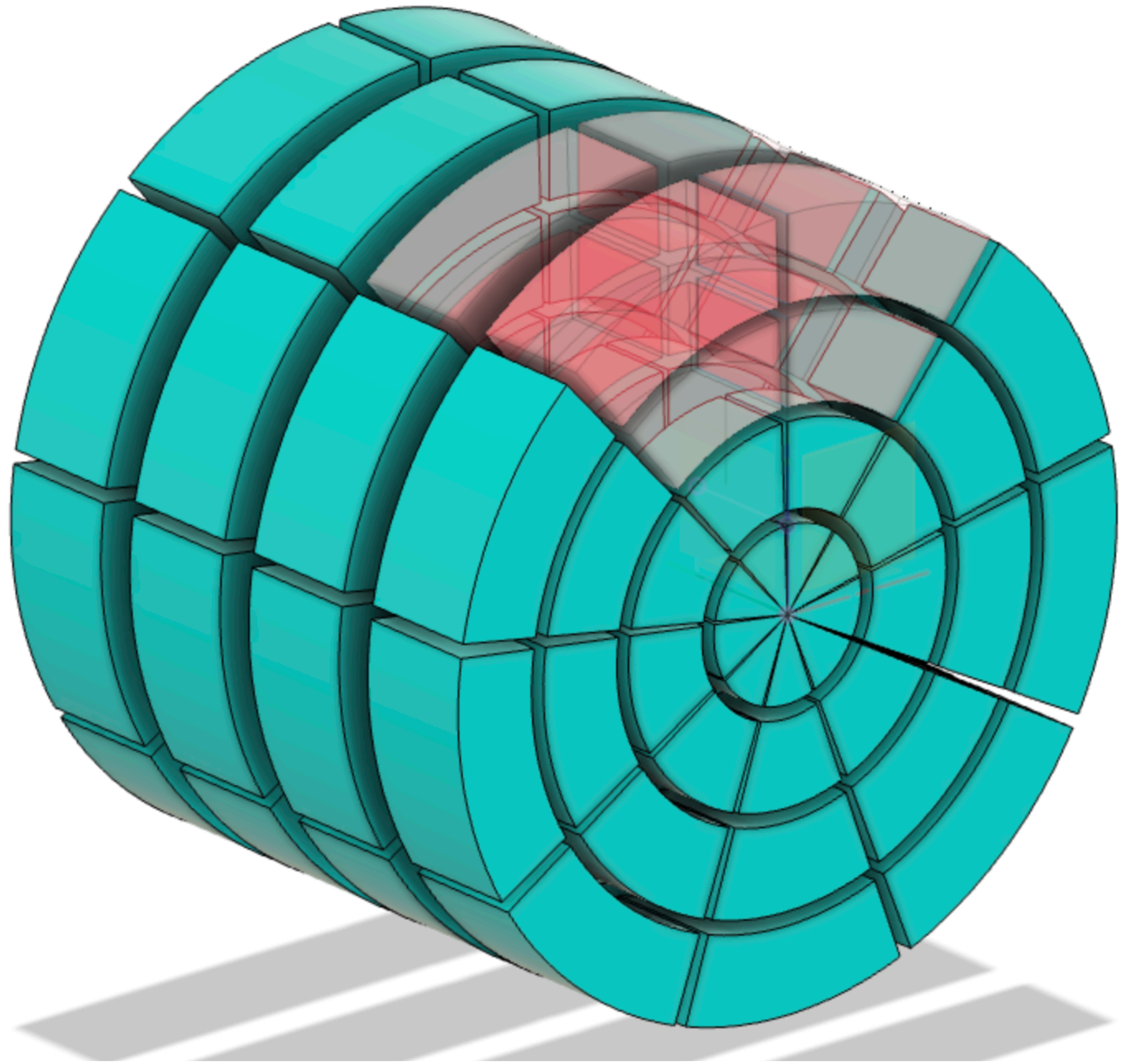


Cylindrical Convolution* (First Approach)

**Strictly, not a real convolution since these shifts are not a homogenous space…*

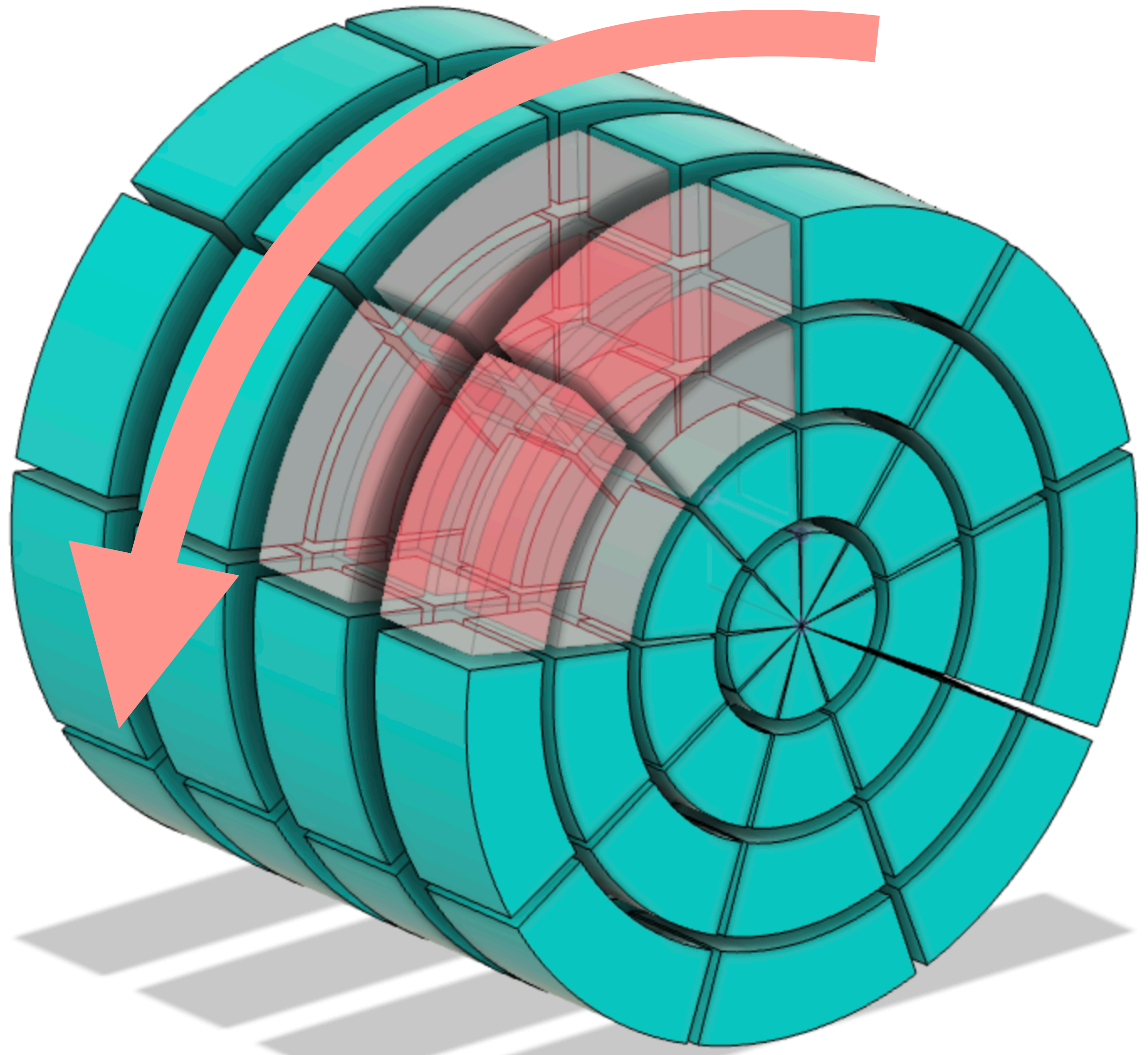
Recall: Matrix convolution is done by shift -> product -> sum

$$[f \star g]_{ij} = \sum_{mn} f_{mn} g_{(i-m),(j-n)}$$

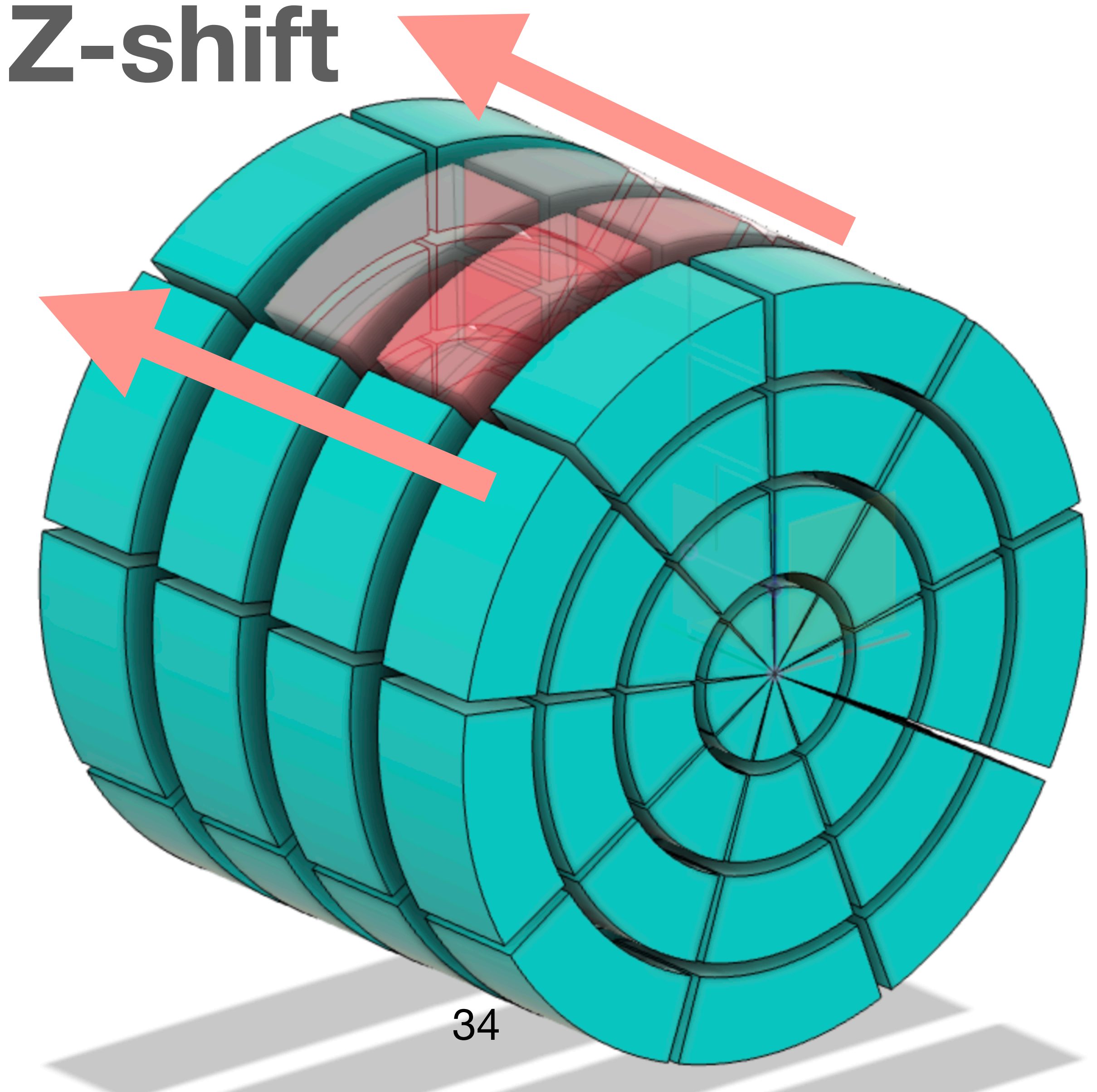


We just have to redefine the shift* operators, and ensure boundary conditions

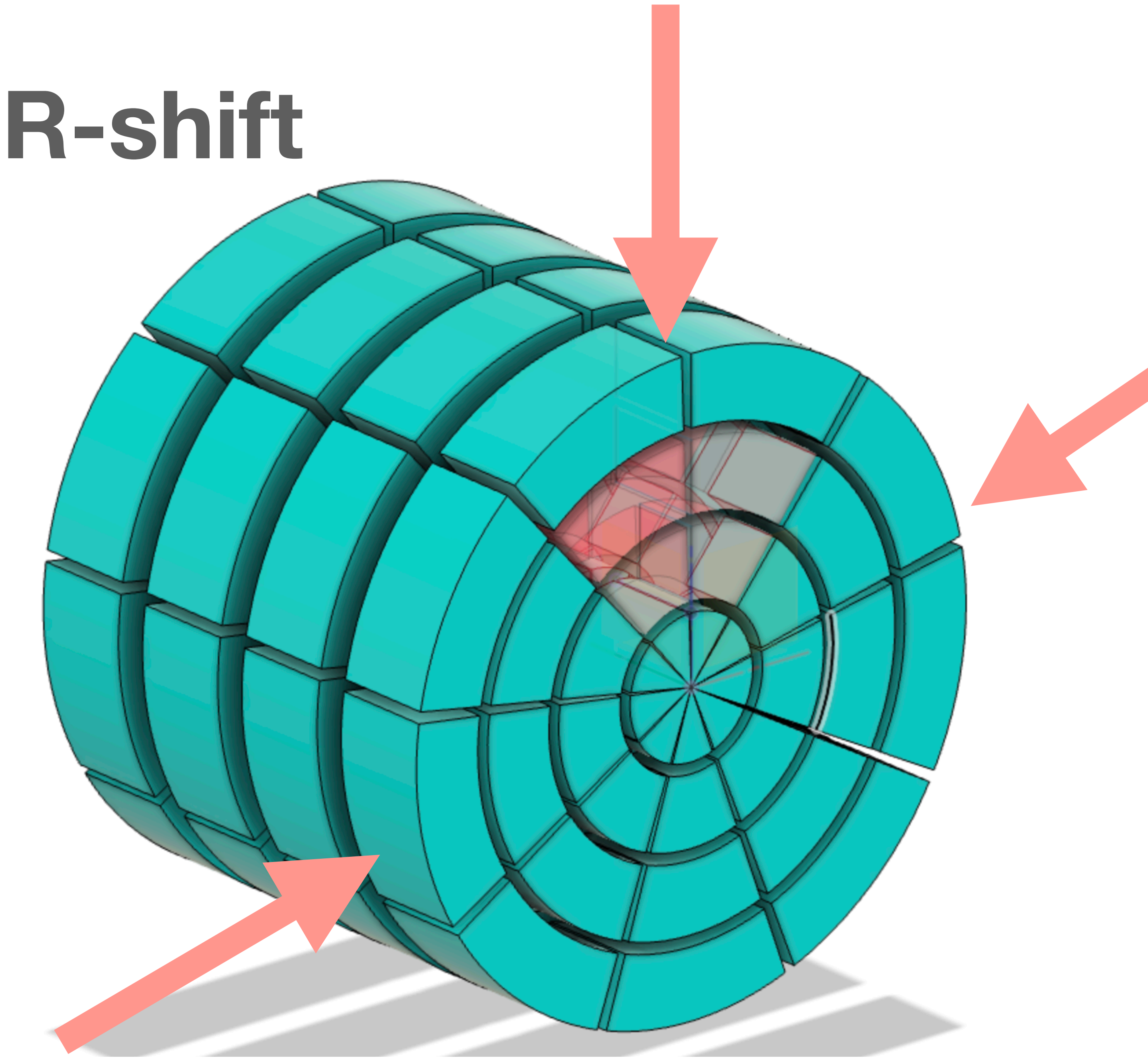
Phi-shift



Z-shift

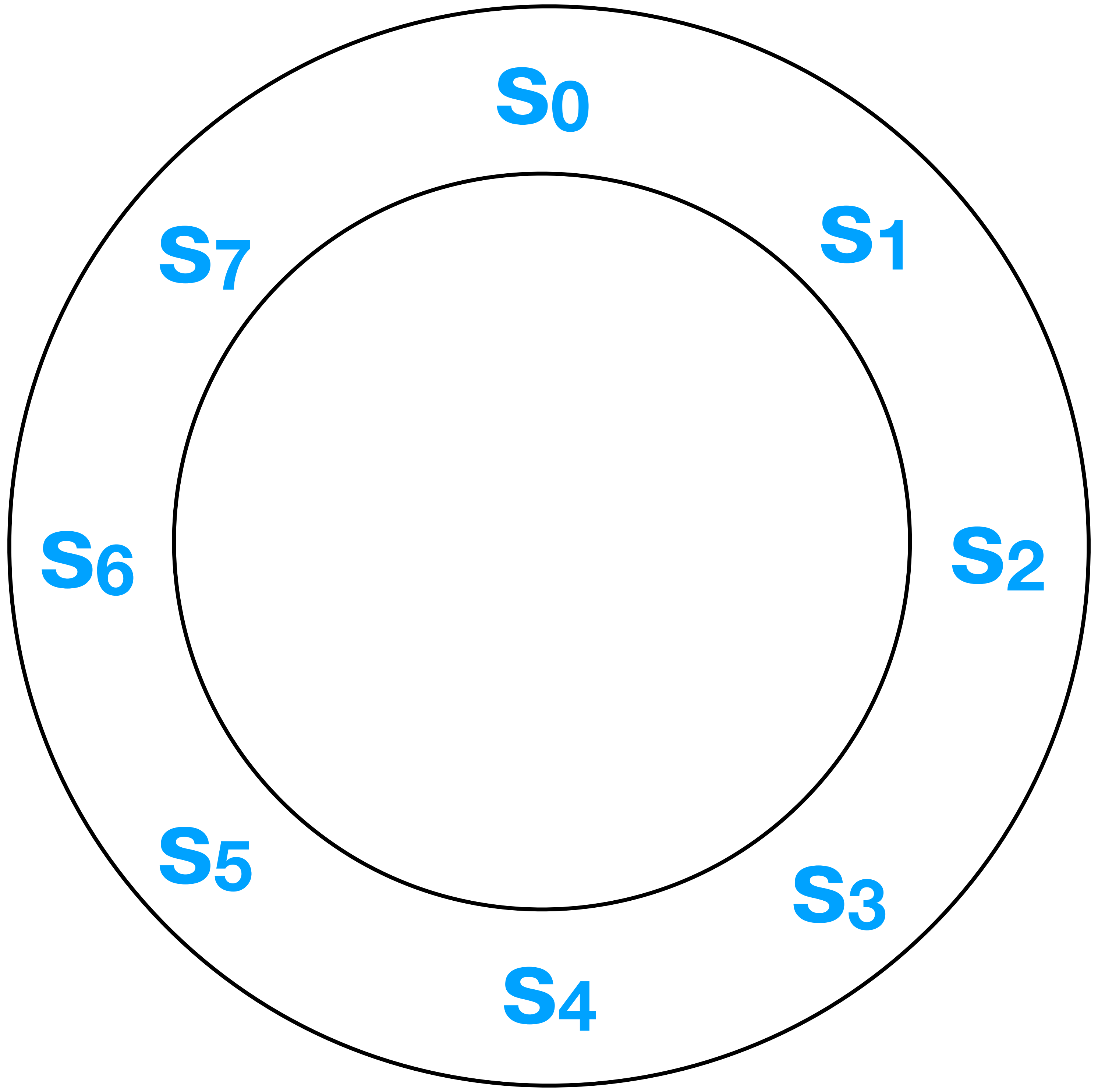


R-shift



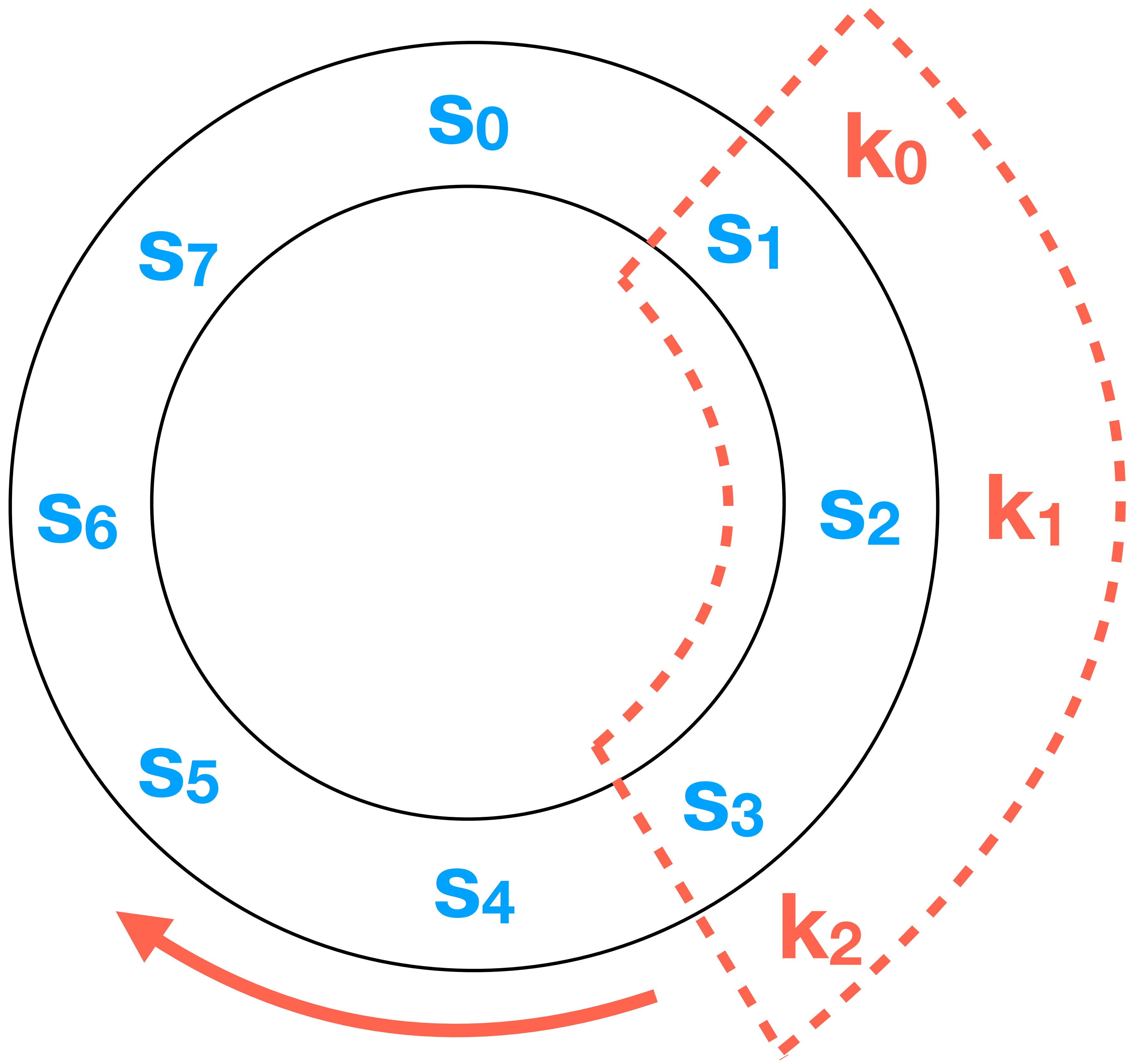
1D Example: Circular Convolution

Consider a **signal s**, sampled on a circle:



1D Example: Circular Convolution

Consider a **signal s**, sampled on a circle:

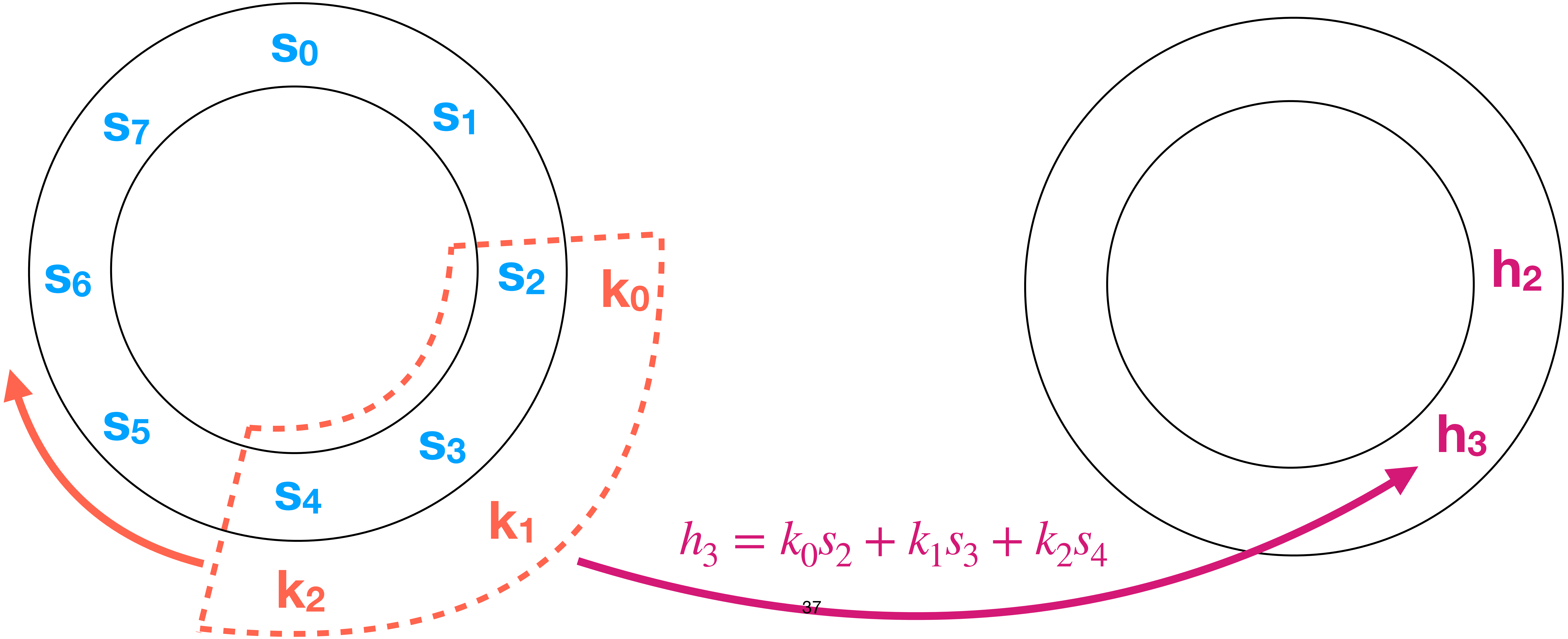


Want to do circular convolution with a **kernel k**

1D Example: Circular Convolution

Consider a **signal s**, sampled on a circle:

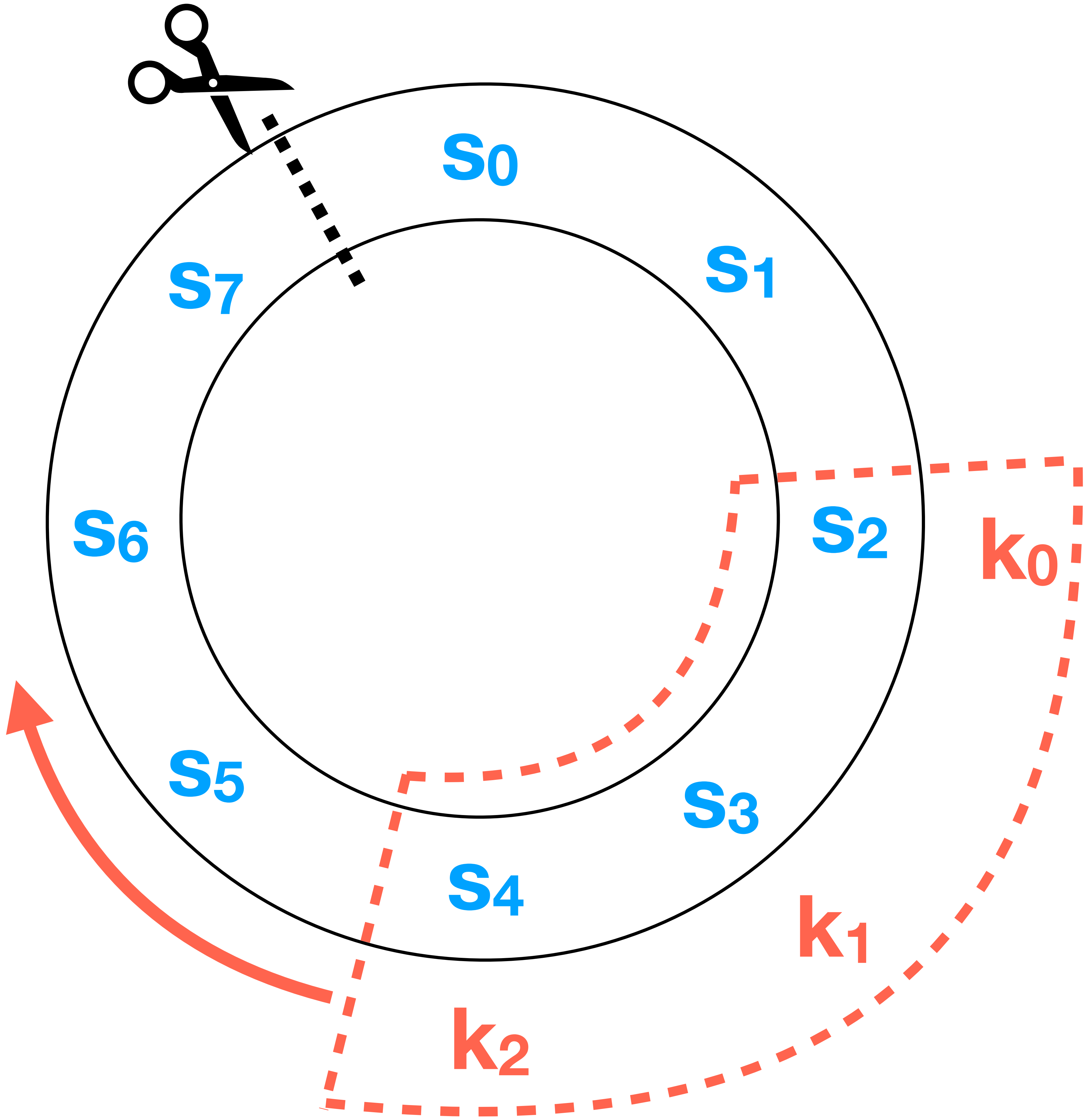
Shift / product / sum to get **output signal h**:

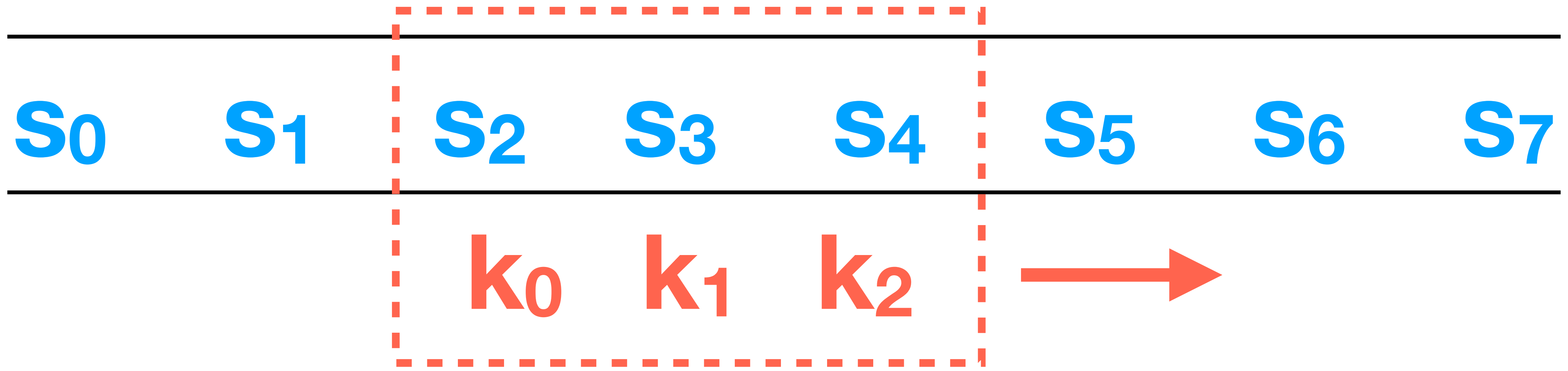


$$h_3 = k_0 s_2 + k_1 s_3 + k_2 s_4$$

1D Example: Circular Convolution

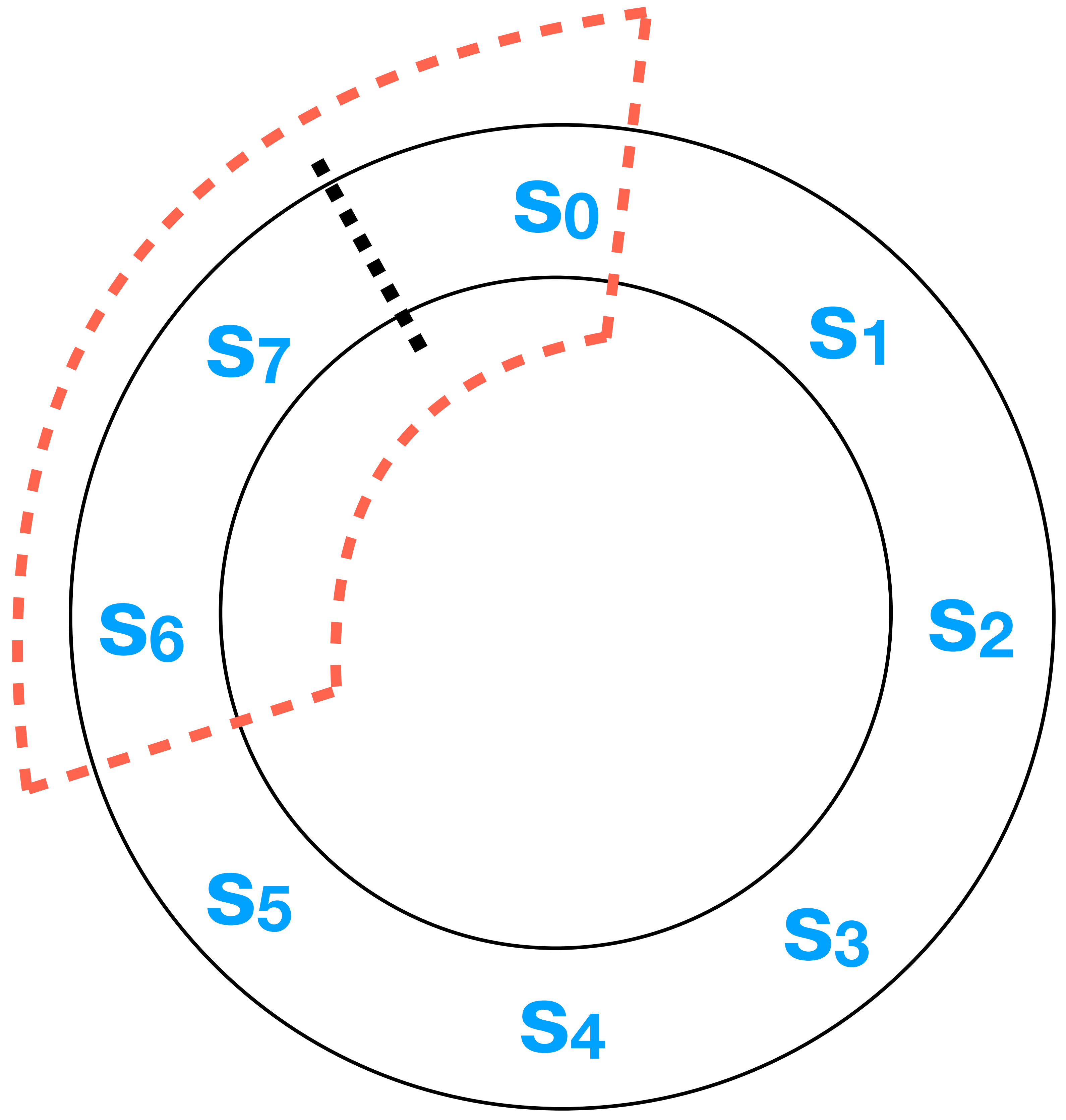
Practical implementation: **unroll the signal** -> Reduce to **normal 1D convolution**!

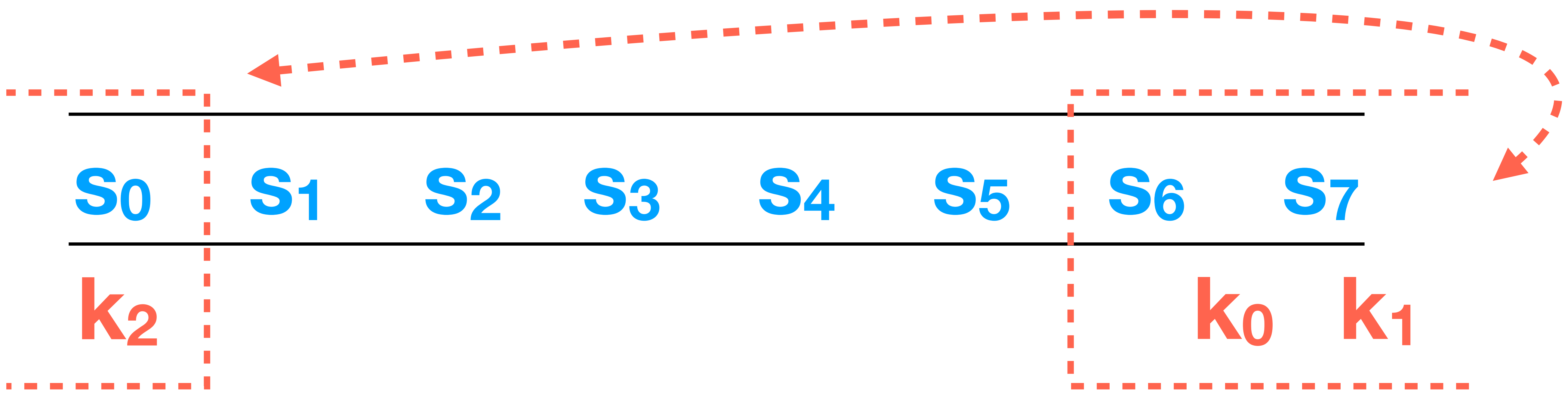




1D Example: Circular Convolution

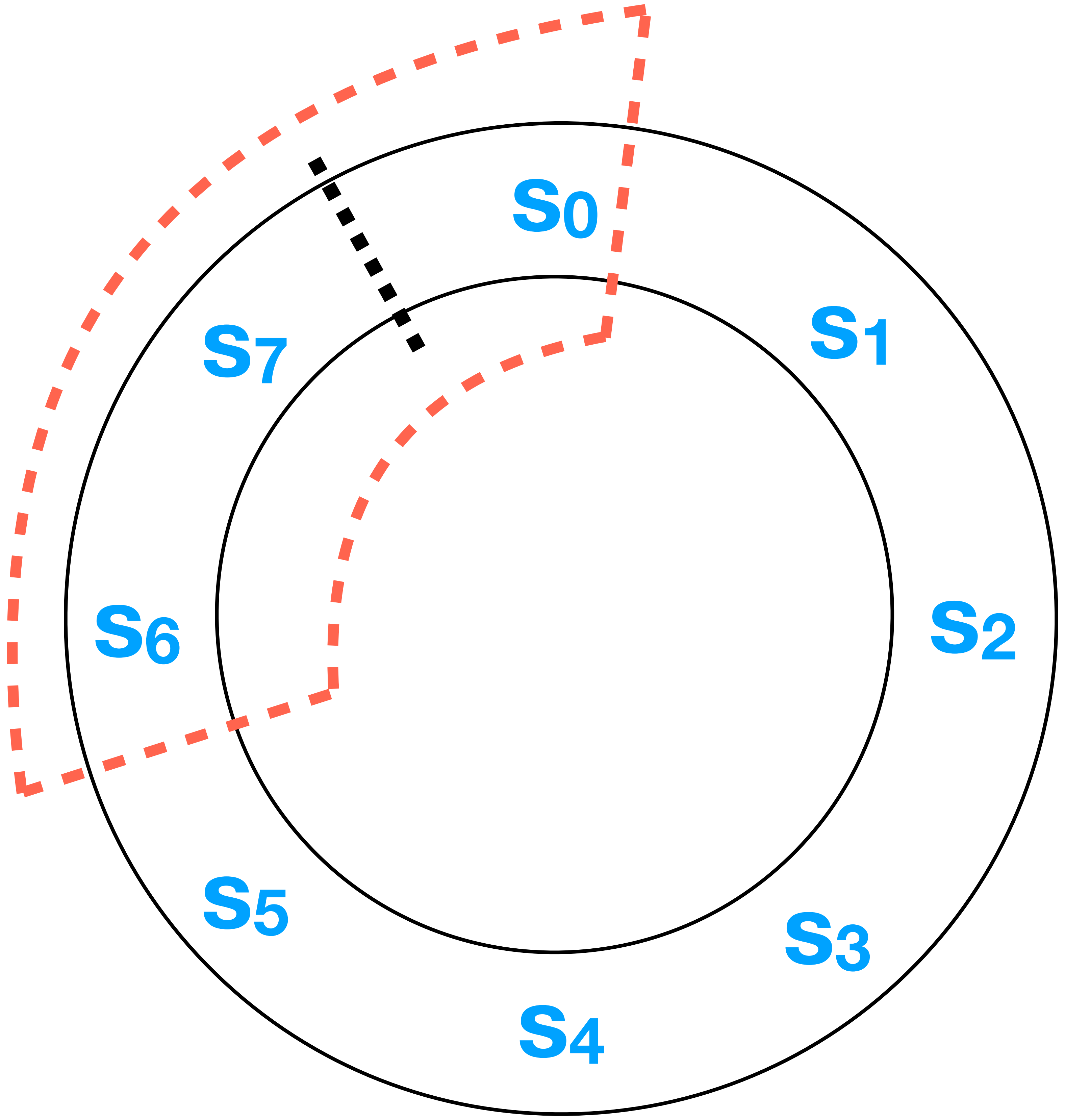
Practical implementation: **unroll the signal** -> Reduce to **normal 1D convolution**!

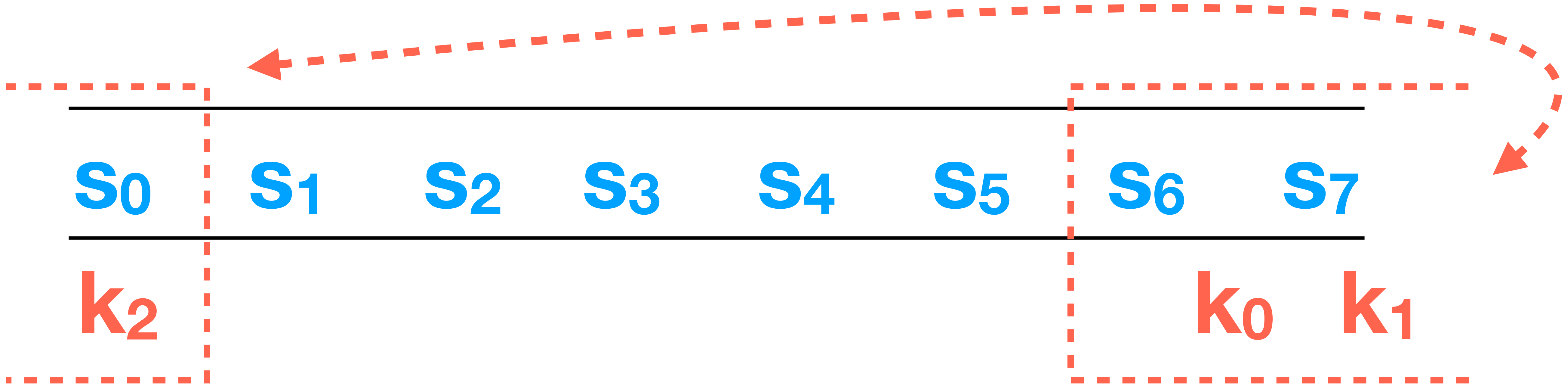




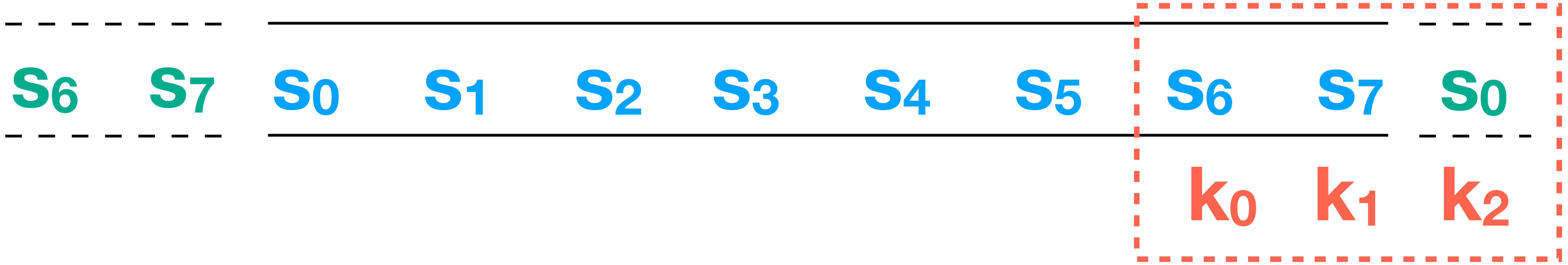
1D Example: Circular Convolution

Practical implementation: **unroll the signal** -> Reduce to **normal 1D convolution**!

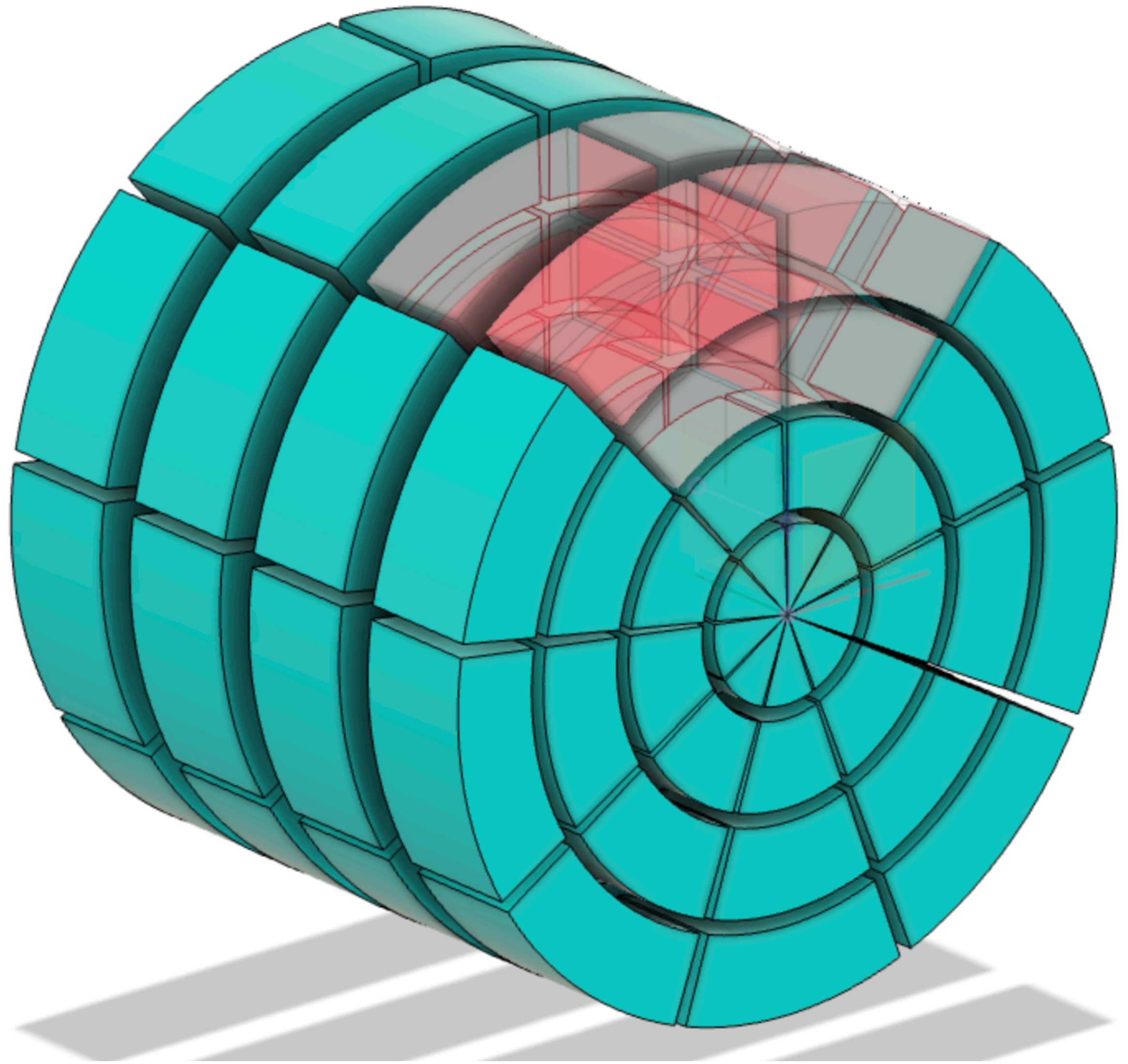




Boundary condition just requires some **careful padding**:



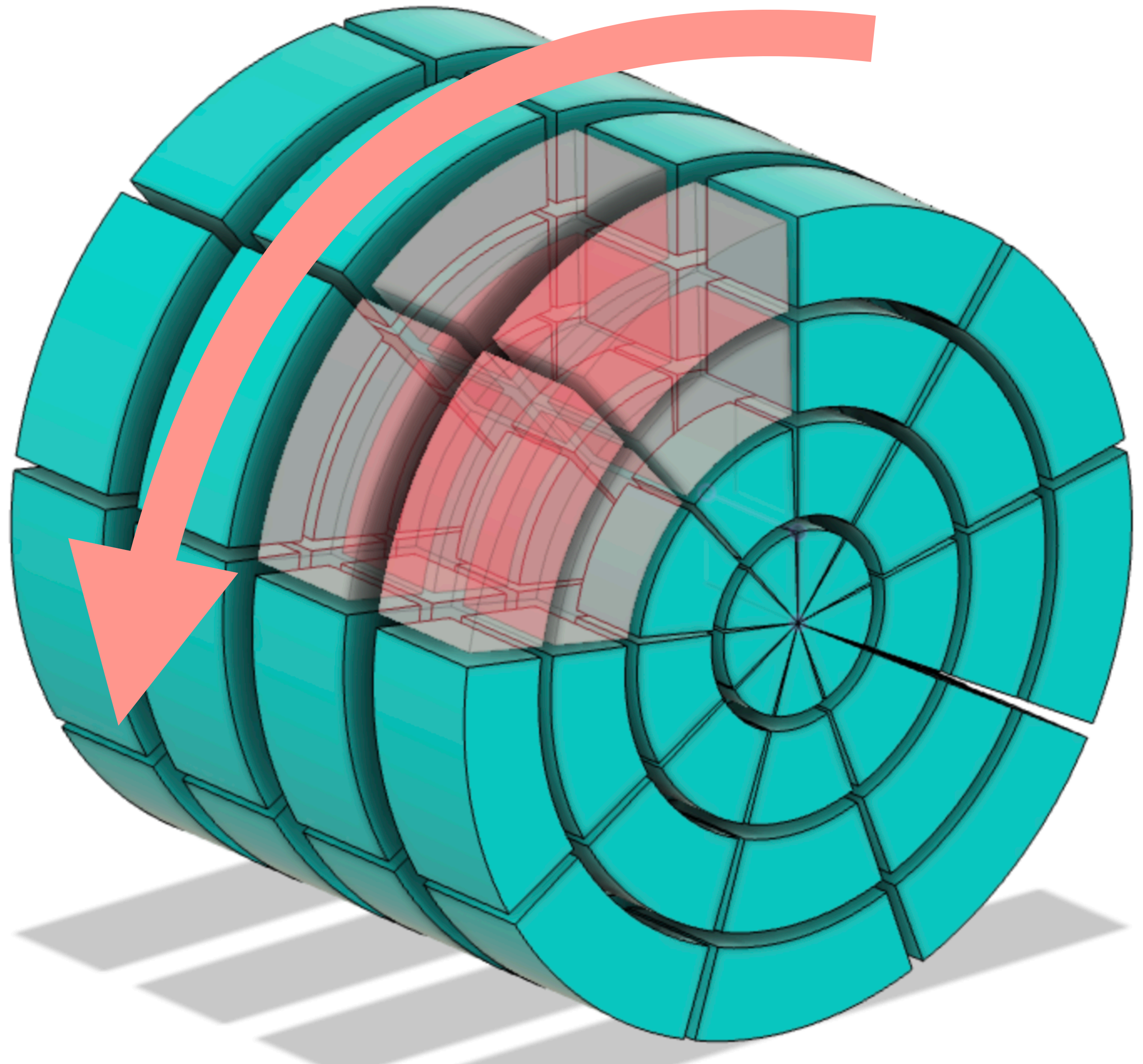
Cylindrical Convolution: Details



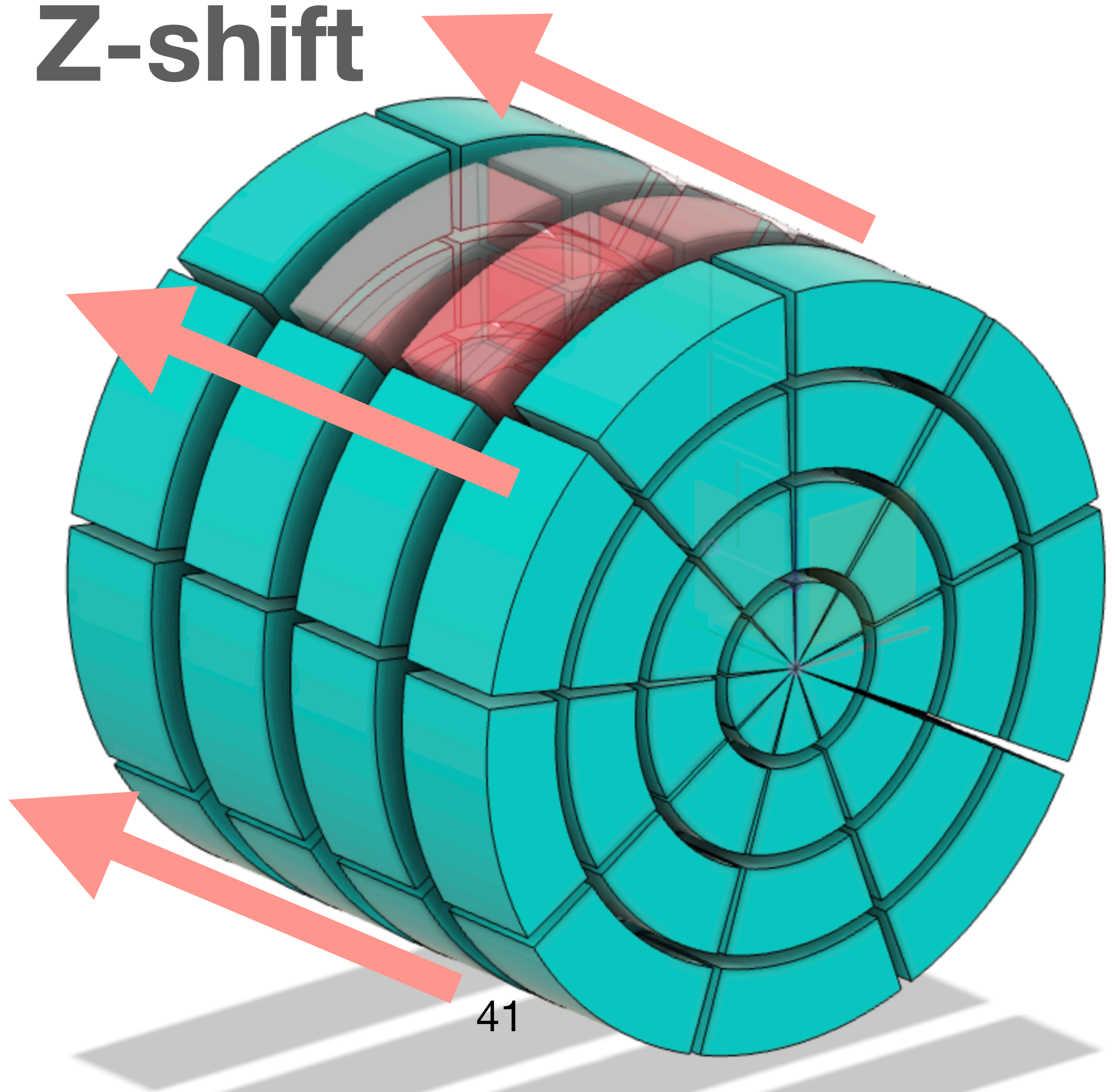
Cylindrical convolution is similar to the 1D example,
we use padding **reduce the problem to standard 3D convolution**.

Can implement other common features such as **kernel strides**, input padding, and transposed convolution.

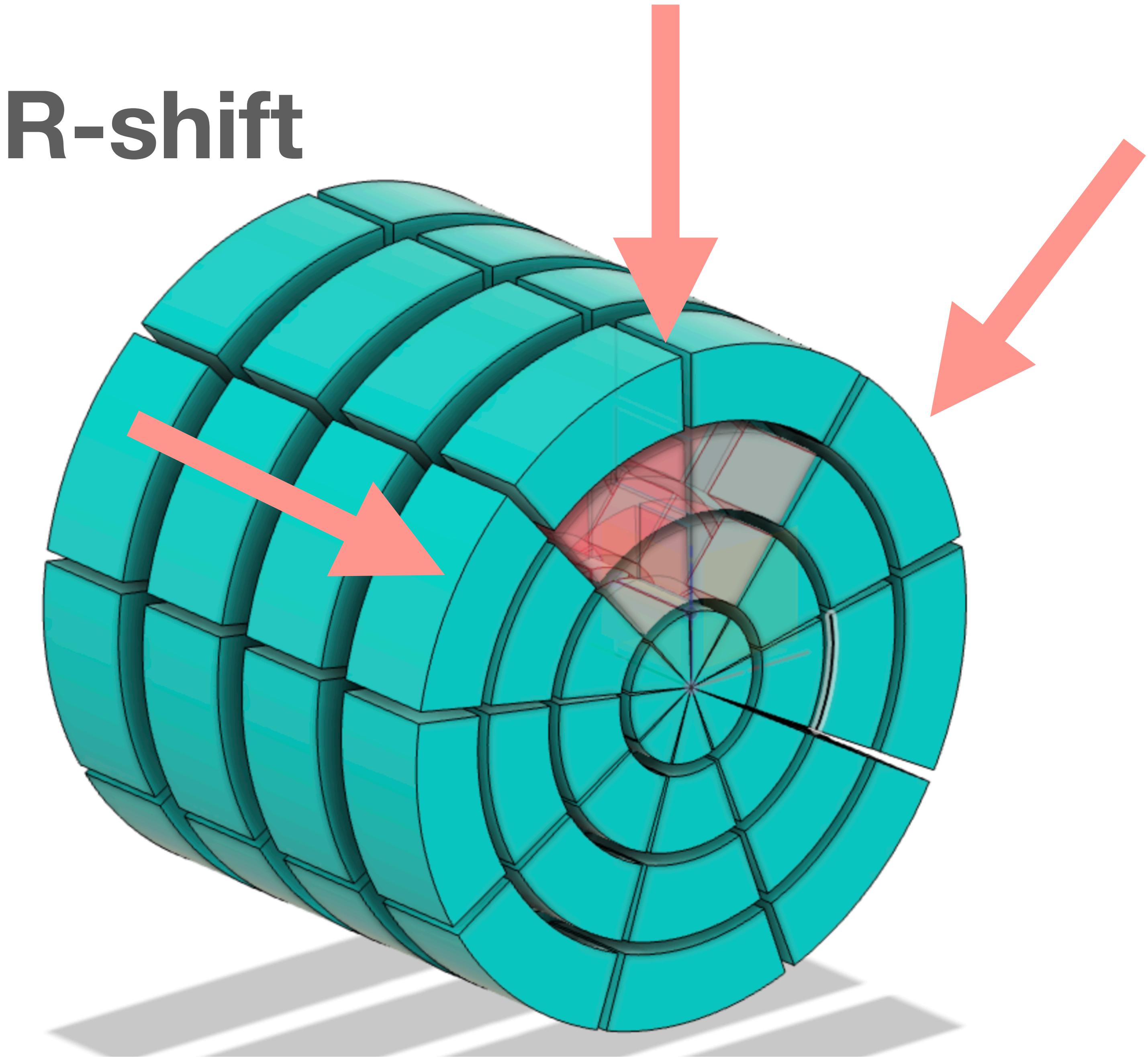
Phi-shift



Z-shift



R-shift

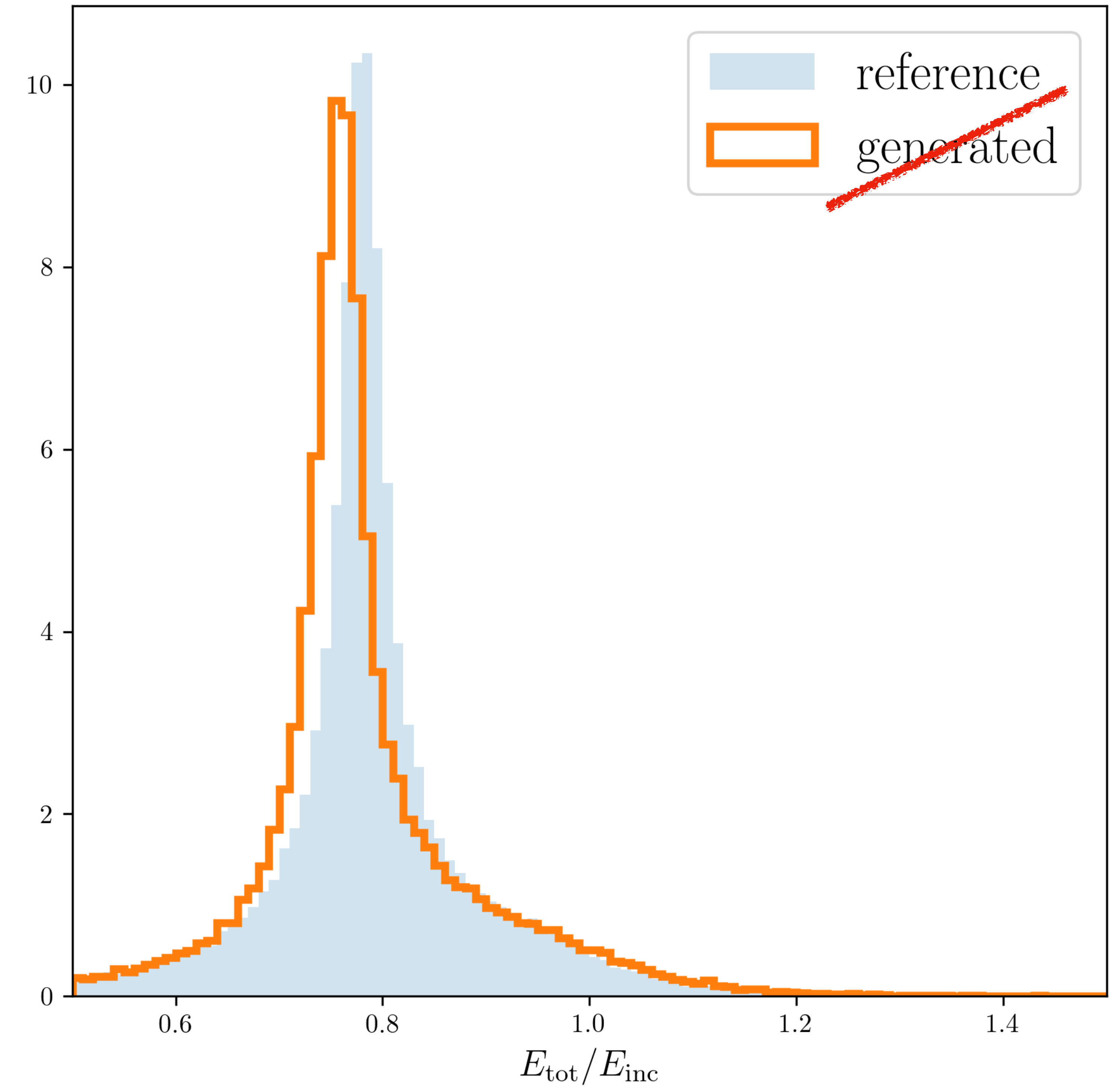


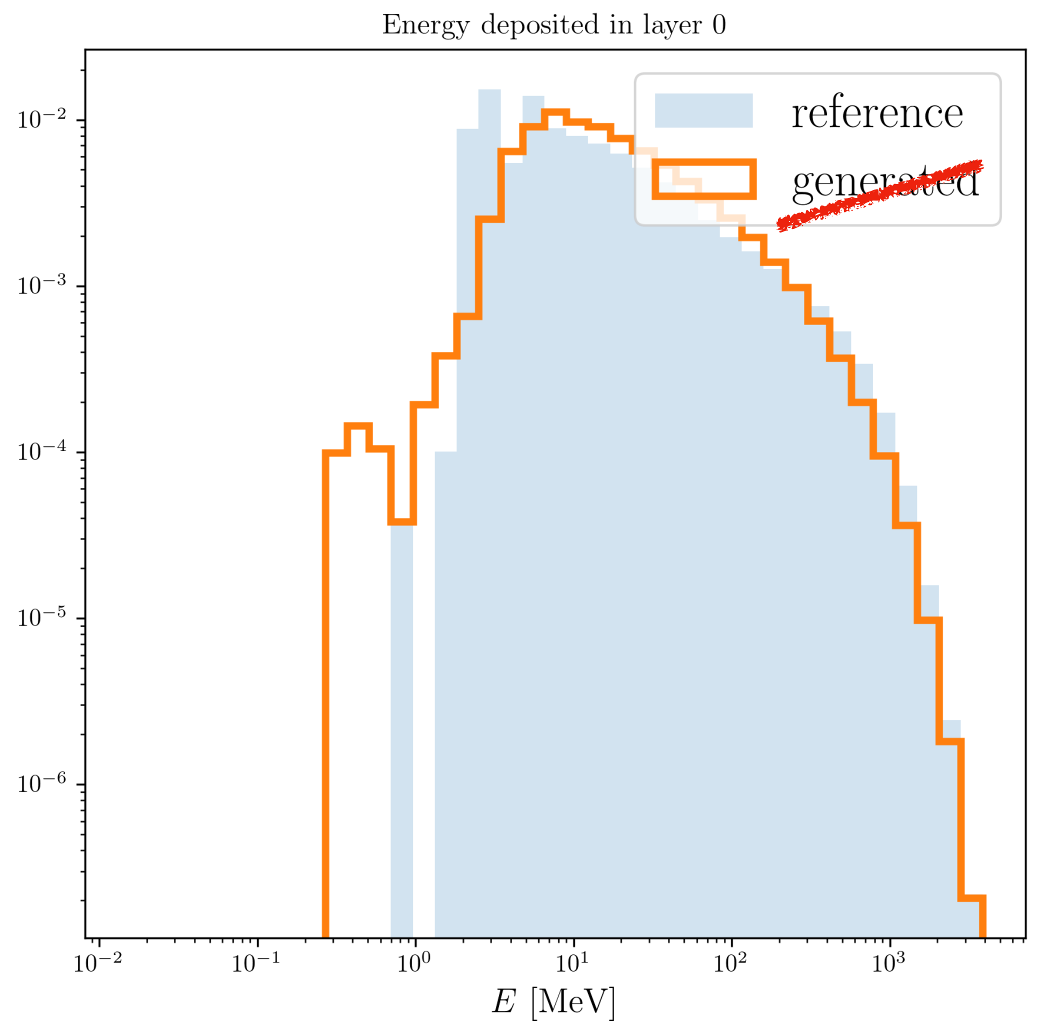
(Yes, it’s cheating)

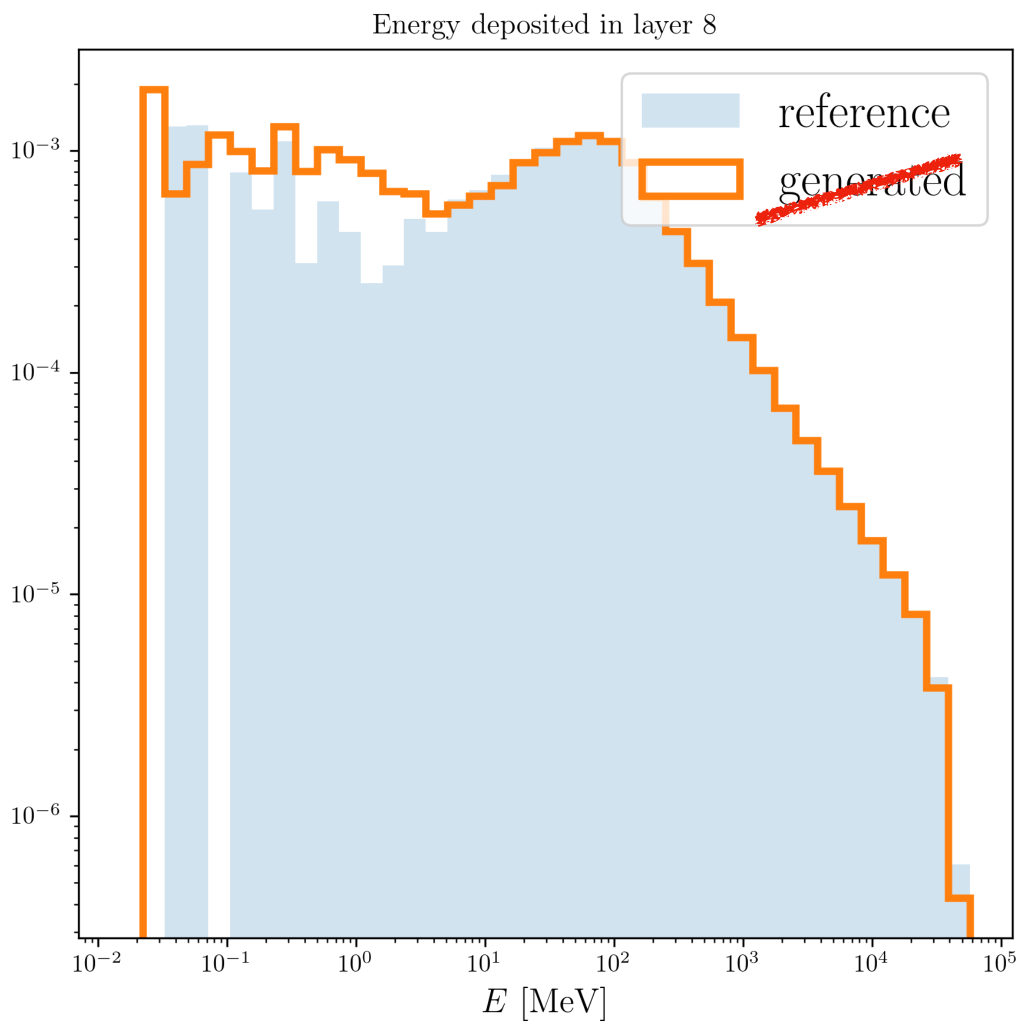
DS 2: Reconstruction Results

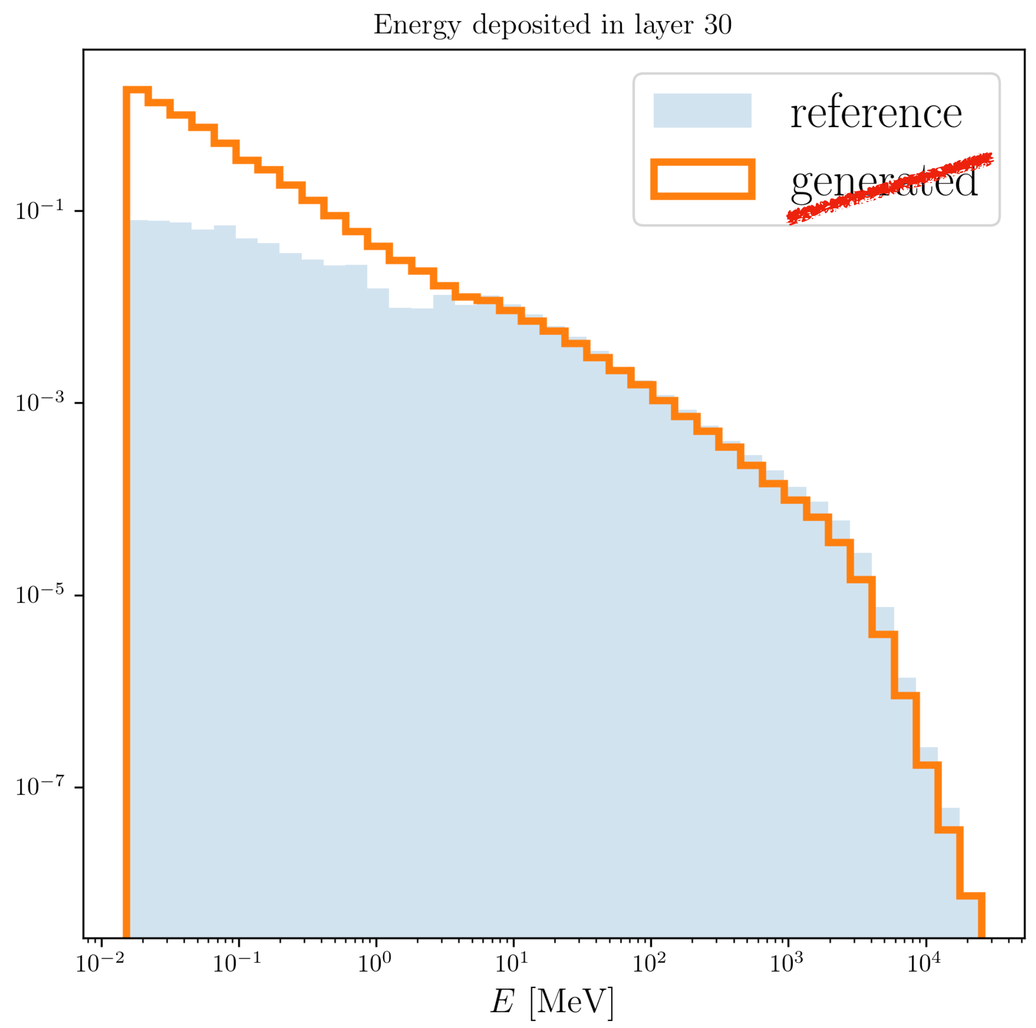
Dataset 2: Energy Metrics

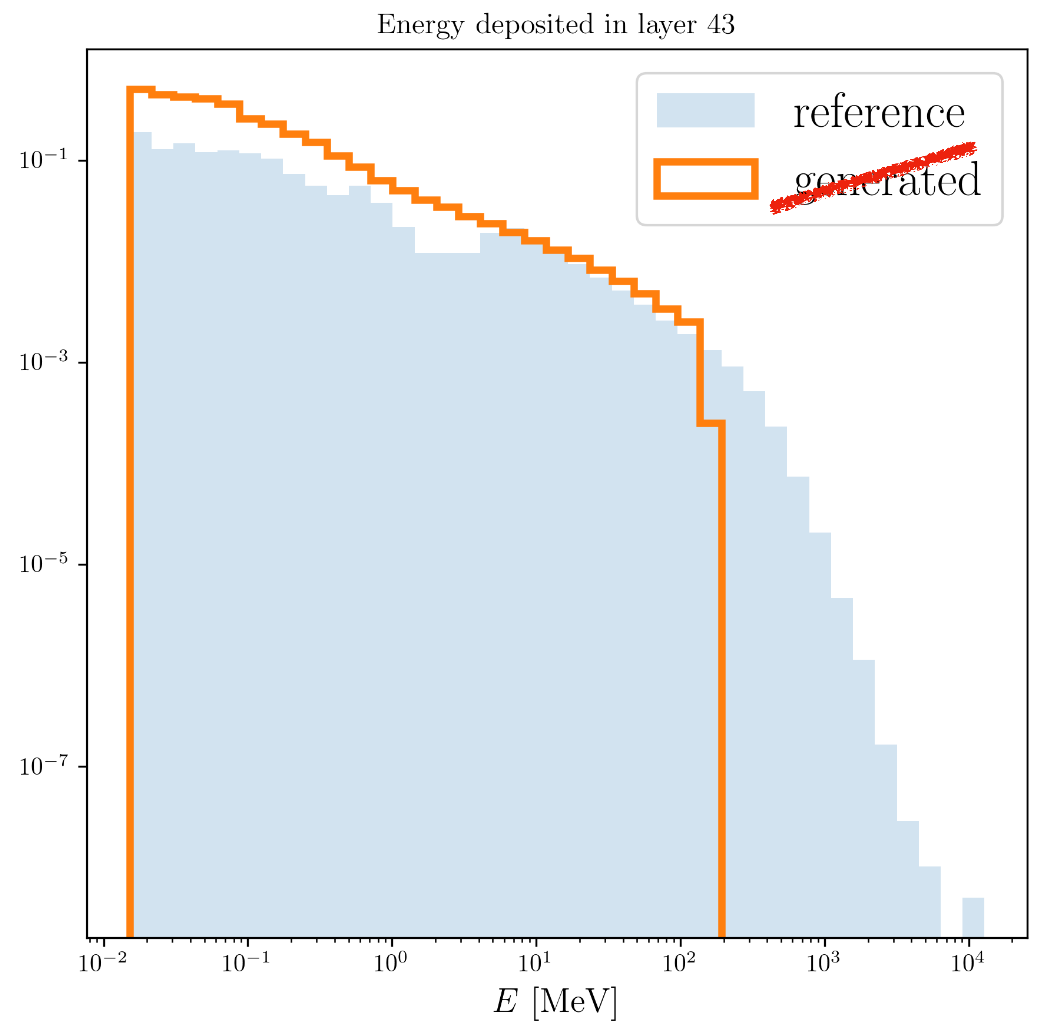
- Results shown for a 3-layer encoder/decoder
- 28 ~> 40 filters per layer
- Latent dimension: **416** (~ 90% spatial compression)
- <150k parameters each for enc/dec



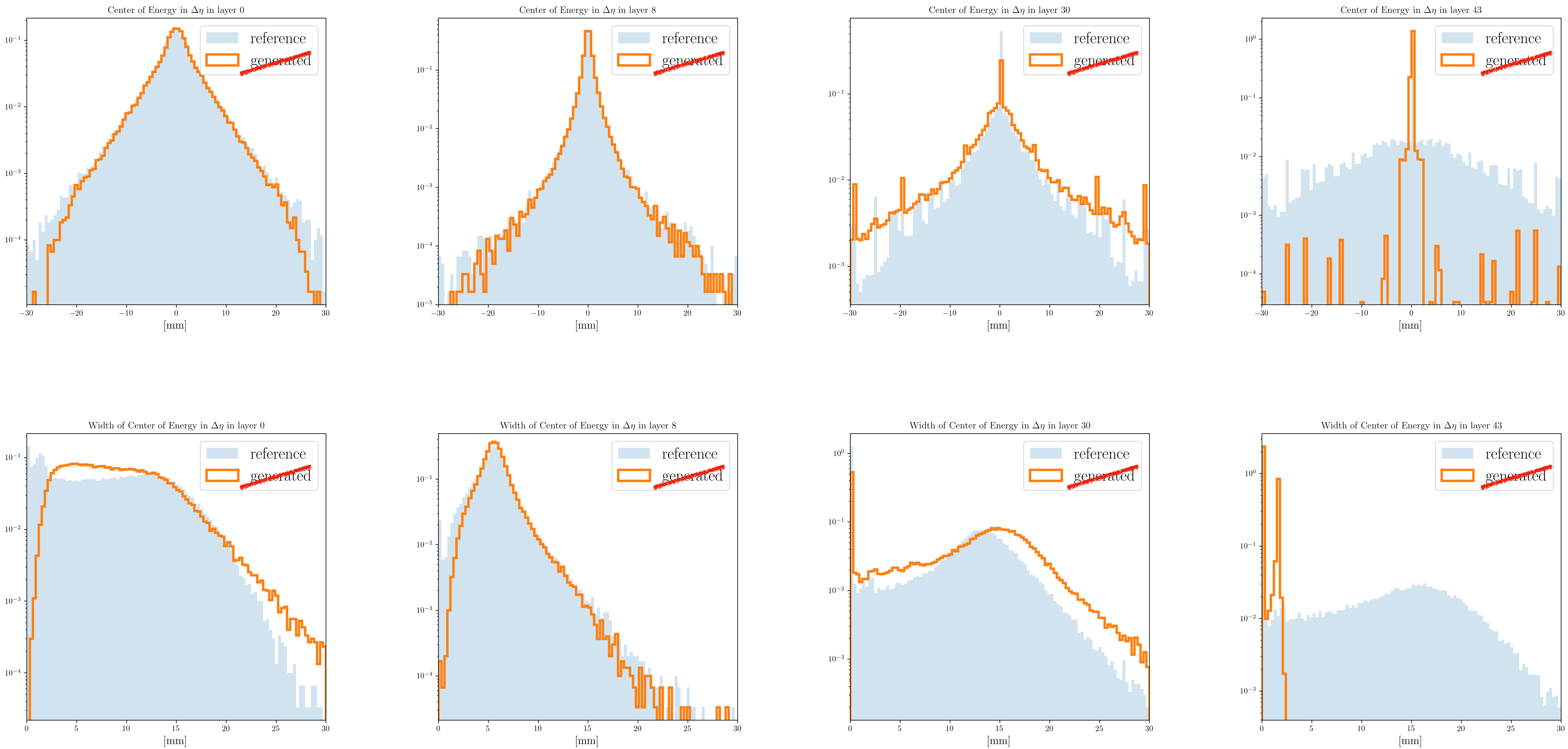








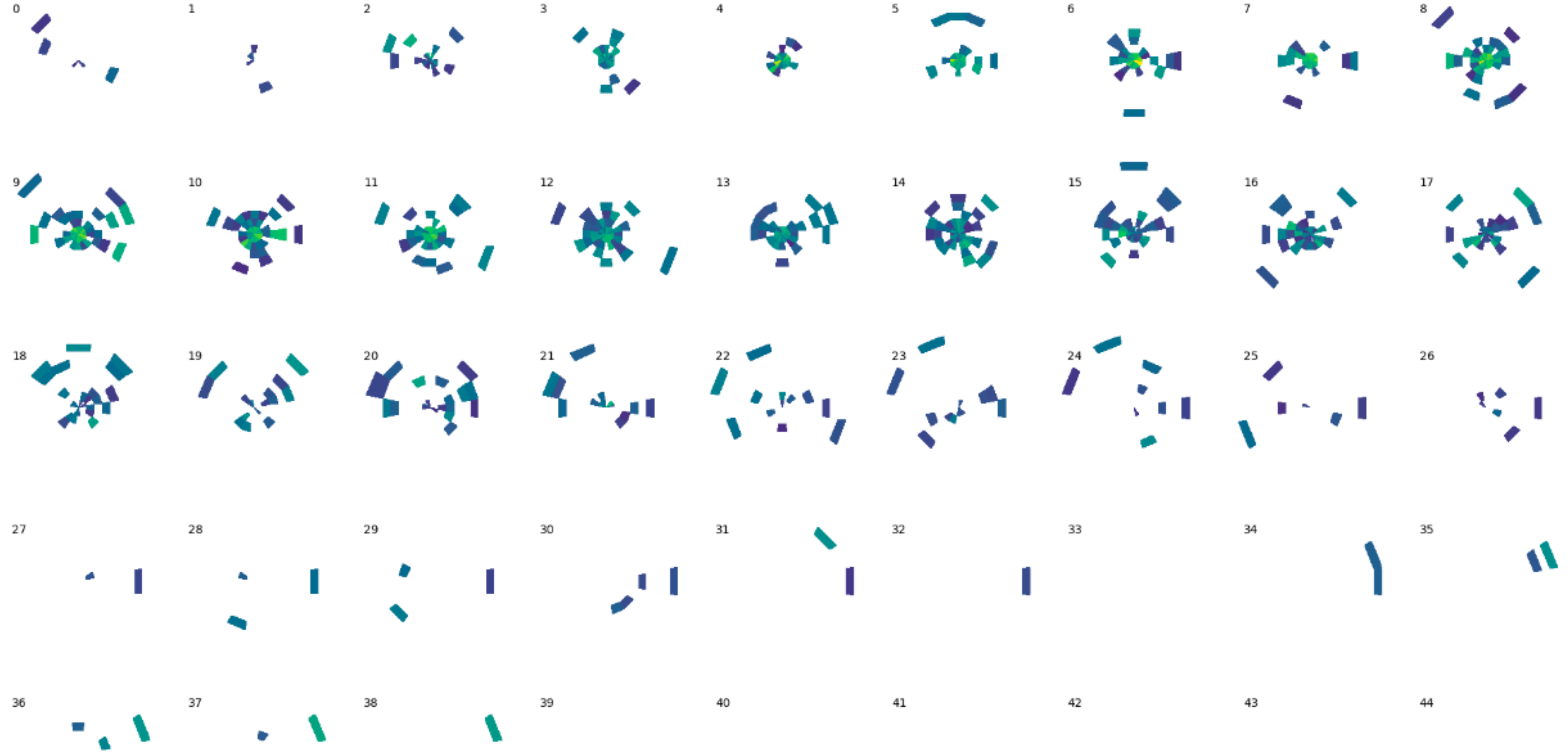
Dataset 2: Shape Metrics



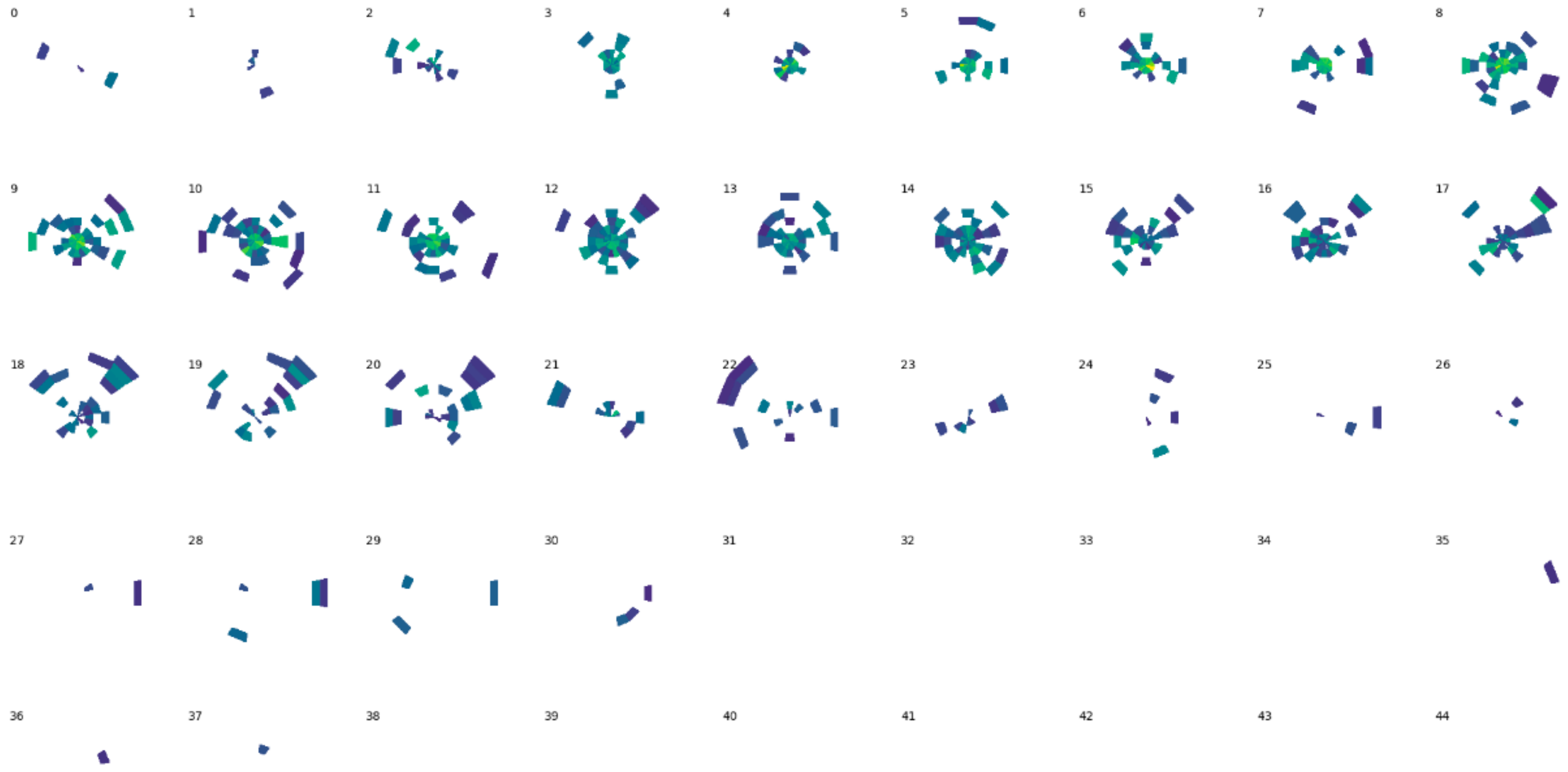
Dataset 2: Event reconstruction

Dataset 2: Single event reco

Truth

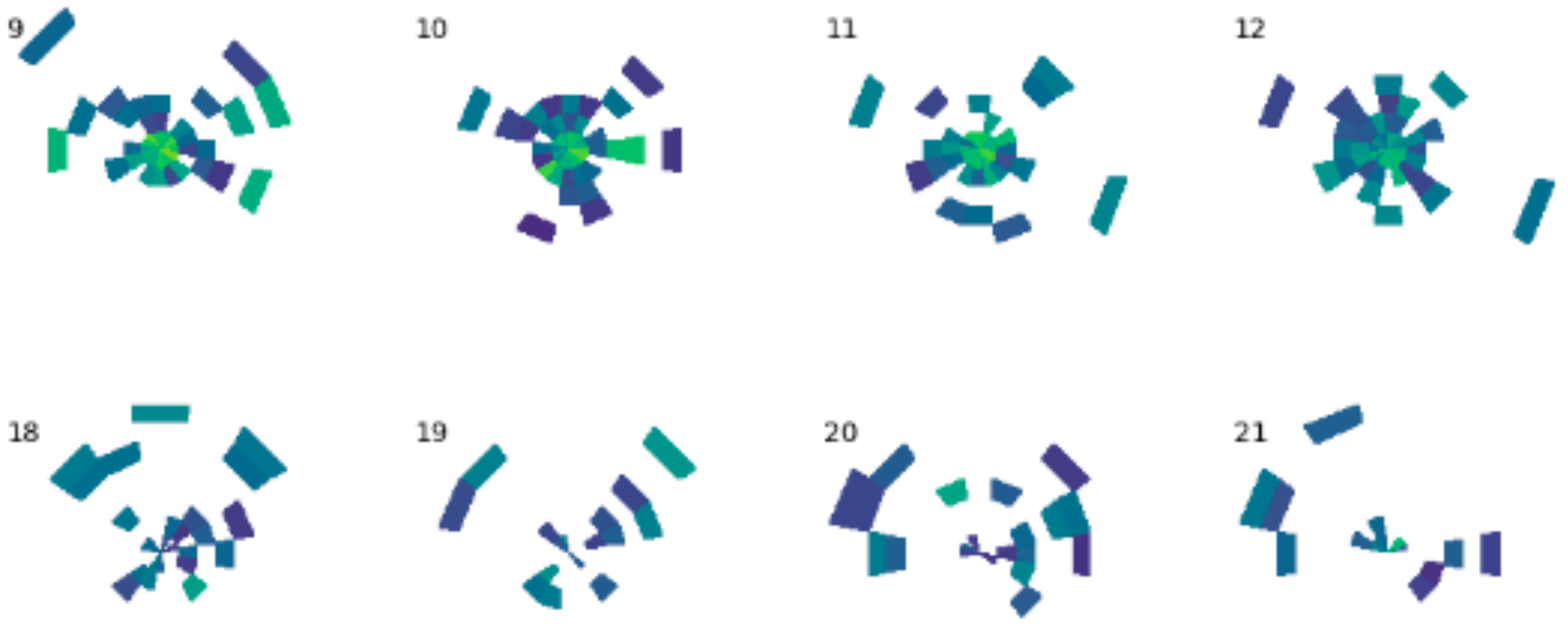


Reconstructed

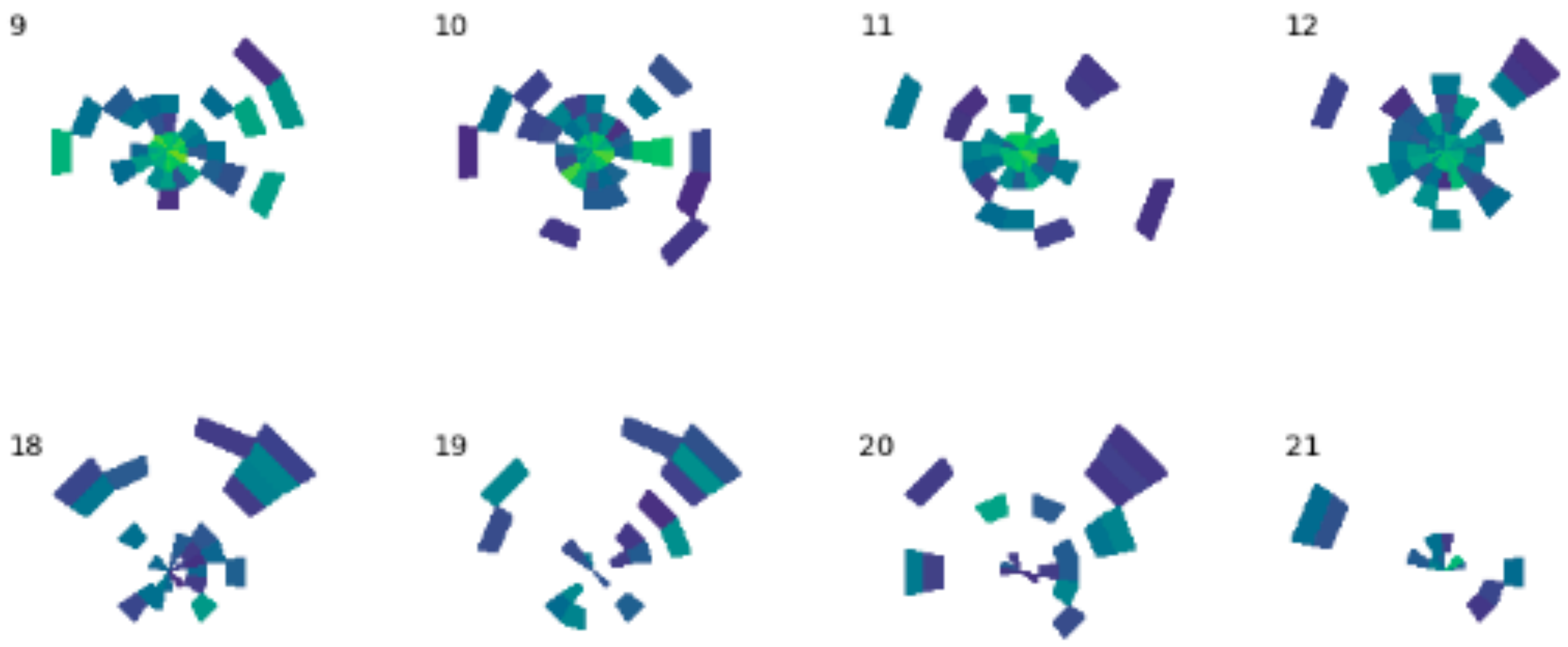


Dataset 2: Single event reco

Truth (detail)



Reconstructed (detail)



Conclusions

Dataset 1:

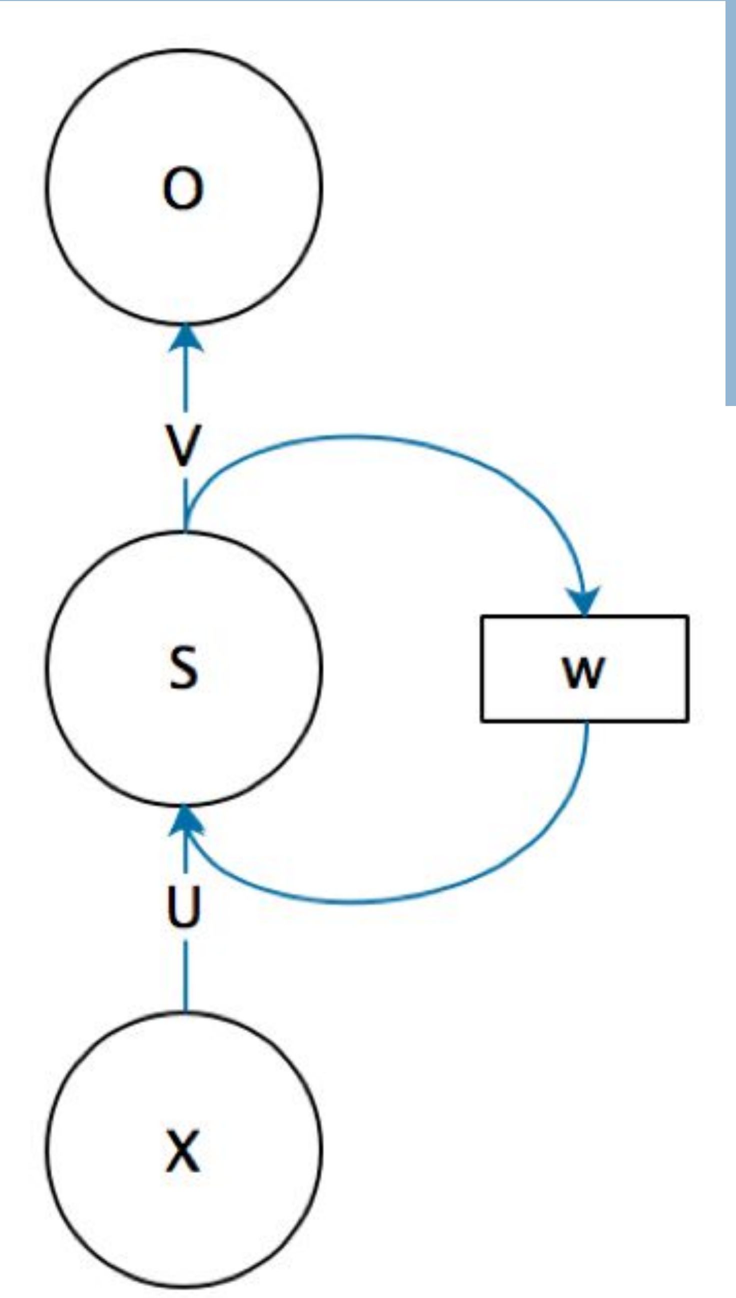
- VQ-VAE enables a two step model with high model compression
- Performance on DS1 is comparable to CaloGAN (but no GAN required!)
  - Both in terms of performance and generation time
  - But VAE can’t compete with CaloFlow! -> will focus on studying VQ-VAE scaling to the larger datasets

Datasets 2+3:

- We plan to further improve the Cylindrical Conv. architecture
  - Factorized model allows us to focus only on reconstruction for now!
- Latent generative model for the convolutional VQ-VAE: **TBD**
  - Looking forward to hearing your ideas!!

Thank you!

PRIORI GENERATION: RNN



- Generate one step based on previous history of steps
  - Compared with DNN: output only depend on current step
- Widely used in generation of sequence like time sequence, text, so on
- Suitable to generate the quantized number and model the joint distribution of discrete choice
- One example to generate 1 3 2 4
  - 0 0 0 0 → 1 : predict next step based on 0 0 0 0, generative seed
  - 0 0 0 1 → 3 : predict next step based on 0 0 0 1
  - 0 0 1 3 → 2
  - 0 1 3 2 → 4