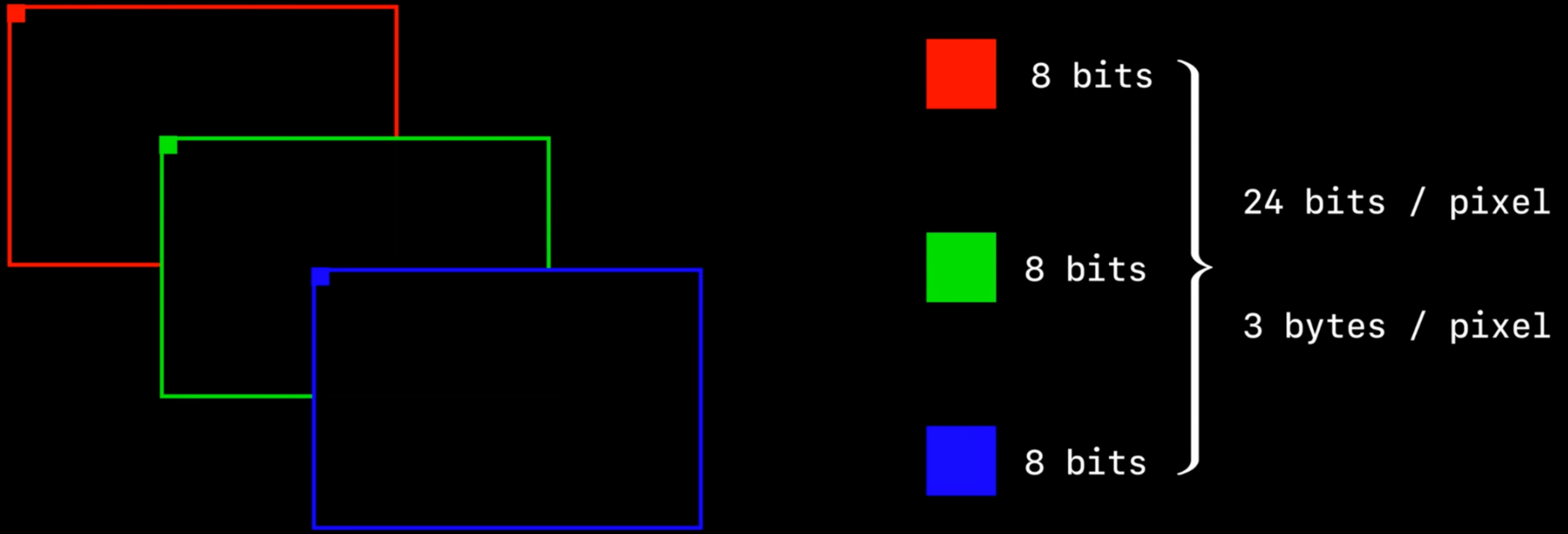
Learned Image Compression

L16, EE274, Fall 23

Slides Credit: Kedar Tatwawadi

Recap: What is an image?

Array of pixels: (Height, Width, Channels)



Recap: Image Compression

764x512x3 bytes
= 1.1MB!
(Uncompressed)

Image from Kodak dataset

Recap: Image Compression -> JPEG 40x

Image from Kodak dataset

Uncompressed -> 1.1MB
JPEG -> 27KB (~40X!)

Recap: Image Compression -> JPEG 80x

Uncompressed -> 1.1MB
JPEG -> 14KB (~80X!)

Image from Kodak dataset

Recap: Image Compression -> JPEG 137x

Uncompressed -> 1.1MB
JPEG -> 8KB (~137X!)

Image from Kodak dataset

Recap: Image Compression -> BPG

Uncompressed -> 1.1MB
BPG -> 8KB (~137X!)

Image from Kodak dataset

Quiz Q1

Q1 Image Compression
4 Points

Before the next big game, facing an inevitable loss, Berkeley students hacked into Stanford website and tried to mutilate the Stanford logo into a Berkeley blue color version (but did a bad job at it). The mutilated logo is shown as an image above.

This image is of dimensions 370×370, and contains 4 channels (RGBA) instead of 3 channels for colors we saw in class. The fourth channel is alpha channel which tells the transparency of the image. The bit-depth of this image is 8, which basically implies that every pixel in each channel is 8 bits.

This file can be compressed losslessly using PNG to ~ 14.3 KB (kilo-bytes).

Q1.1 What's the expected raw size of this image?
Ans: 370 x 370 x (24+8) **bits** = 370 x 370 x (3+1) **bytes** = 547.6 KB

Q1.2 In this image you can see that there are basically just two colors (white and a bad version of Berkeley blue color). What will be the expected image size if we use only 2 colors to compress this image in KB?
Ans: 370 x 370 x (1+8) **bits** = 154.012 KB

Q1.3 Now you also see that along with having just 2 colors, the image also has only two levels of transparency (perfectly transparent and perfectly opaque). Using these properties what will be the expected image size in KB?
Ans: 370 x 370 x (1+1) **bits** = 34.225 KB

Q1.4 PNG seems to perform better than even using 1 bit for color and 1 bit for alpha!
Ans: Better entropy coding!
PNG performs better because it also uses a version of **LZ77** to look for matching pixels and store match length/offset pairs, rather than each pixel individually. Because this image has a lot of redundancy, there are probably many matches LZ77 can identify.

Quiz Q2

Q2 JPEE274G Compressor
4 Points

EE274 students decided to come together and form JPEE274G (Joint Photographers EE 274 Group) coming up with an image compressor with the same name. Help them make the design decisions.

Q2.1
2 Points

Riding on the compute revolution, JPEE274G decided to go for 64×64 block size instead of 8×8.

Suppose you have the same image at resolution 480×480, 720×720, 1080×1080

In which of the following case do we expect increasing the block-size help the the most.

- ○ 480×480
- ○ 720×720
- ◉ 1080×1080

Q2.2
2 Points

JPEE274G decided to use prediction of blocks based on previously encoded neighbors. In which of the following two images do we expect the prediction to help the most.

- ◉ Charizard (the one with the orange cranky being)
- ○ Assorted Pokémons (the one with Pokemon written in it)

Ideas:

1. Higher resolution same image => less correlation between neighboring pixels => need to increase block size
2. More homogenous blocks => better predictive coding using previous blocks => more savings

Quiz Q3

Q3 Predictive Coding

2 points

You find a source where consecutive values are very close, so you decide to do predictive lossy compression. Encoder works in following fashion: it first transmits the first symbol and after that it quantizes the error based on prediction from last encoded symbol. The quantized prediction error is transmitted.

Formally, suppose $X_1, X_2, \ldots$ is your original sequence and $\hat{X}_1, \hat{X}_2, \ldots$ is the reconstruction sequence. Then we have:

- for the first symbol the reconstruction $\hat{X}_1 = X_1$, i.e., you are losslessly encoding the first symbol
- prediction for X_n is simply $\hat{X}_{n-1}$
- prediction error is $e_n = X_n - \hat{X}_{n-1}$
- quantized prediction error is $\hat{e}_n$
- reconstruction for X_n is $\hat{X}_n = \hat{X}_{n-1} + \hat{e}_n$
- the transmitted sequence is $X_1, \hat{e}_2, \hat{e}_3, \ldots$

For this question, assume that the quantization for the prediction error is simply integer floor.

Example encoding for source sequence: $0.4, 1.1, 1.5, 0.9, 2.1, 2.9$

n	$\hat{X}_{n-1}$	X_n	e_n	$\hat{e}_n$
1	-	0.4	-	-
2	0.4	1.1	0.7	0
3	0.4	1.5	1.1	1
4	1.4	0.9	-0.5	-1
5	0.4	2.1	1.7	1
6	1.4	2.9	1.5	1
7	2.4	-	-	-

Q1.1 What's the absolute value of the reconstruction error $|X_n - \hat{X}_n|$ for the last symbol?

Ans:

$X_6 = 2.9; \hat{X}_6 = 2.4$ => error = 0.5

Q1.2 Given the transmitted sequence $X_1, \hat{e}_2, \hat{e}_3, \ldots = 1.1, 0, 1, -1, 2, -1$, what is the final decoded value of $\hat{X}_6$?

Ans:

1.1 + 0 + 1 + 0 -1 +2 -1 = 2.1

Recap: Image Compression -> JPEG 137x

Image from Kodak dataset

Uncompressed -> 1.1MB
JPEG -> 8KB (~137X!)

Recap: Image Compression -> BPG

Uncompressed -> 1.1MB
BPG -> 8KB (~137X!)

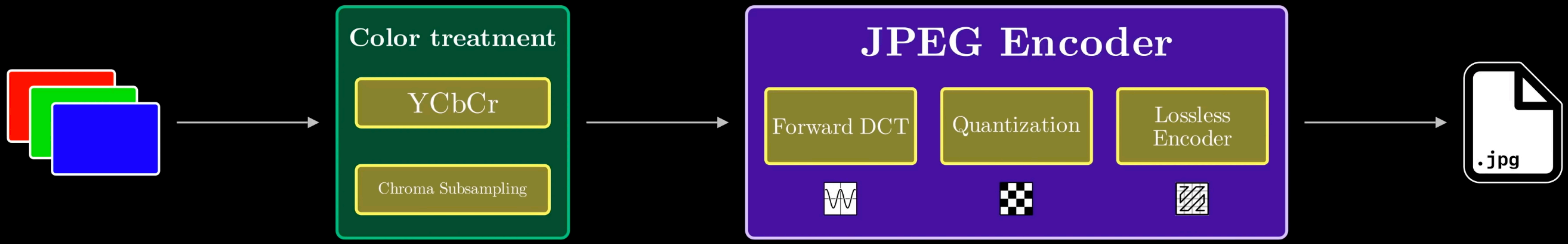
Image from Kodak dataset

Recap: HiFiC -> ML-based image compression

Uncompressed -> 1.1MB
BPG -> 8KB (~137X!)

Image from Kodak dataset

Recap: JPEG



Color treatment
YCbCr
Chroma Subsampling
JPEG Encoder
Forward DCT
Quantization
Lossless Encoder
.jpg

How can we improve further?

Beyond Linear transform: JPEG/JPEG2000/BPG all use variants of DCT, DWT etc. can we obtain better performance with non-linear transforms

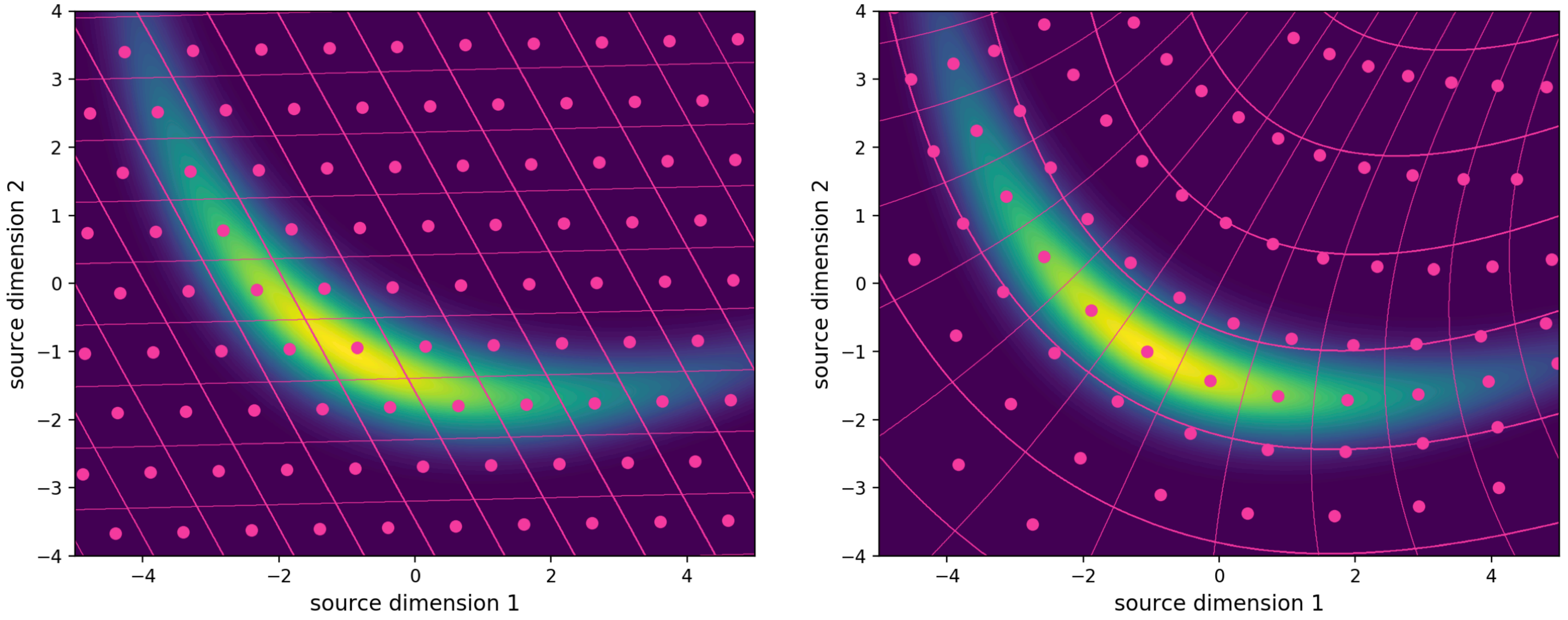


Fig. 1. Linear transform code (left), and nonlinear transform code (right) of a banana-shaped source distribution, both obtained by empirically minimizing the rate–distortion Lagrangian (eq. (13)). Lines represent quantization bin boundaries, while dots indicate code vectors. While LTC is limited to lattice quantization, NTC can more closely adapt to the source, leading to better compression performance (RD results in fig. 3; details in section III).

Ballé, Johannes, et al. "Nonlinear transform coding." IEEE Journal of Selected Topics in Signal Processing 15.2 (2020): 339-353.

What next?

Beyond Linear transform: JPEG/JPEG2000/BPG all use variants of DCT, DWT etc. can we obtain better performance with non-linear transforms

End-to-End RD Optimization: JPEG uses very smart albeit heuristics for R-D optimization, e.g. rate needs to be shared between different channels. Can we make R-D decisions end-to-end?

Recent Learned Image/Video Codec Works

- **Learned Image Compression:**
  [Toderici, CVPR15], [Theis, ICLR17], [Agustsson, NIPS17], [Baig, NIPS17], [Balle, ICLR17], [Rippel, ICML17], [Balle, ICLR18], [Johnston, CVPR18], [Mentzer, CVPR18], [Choi, ICLR19], [Balle, ICLR19], [Lee, ICLR19], [Mentzer, CVPR19] [Lu, 2021], [Ma et al, 2021], [Yang et al, NIPS2021], [Mentzer et al. NIPS2021] . . .

- **Learned Video Compression**
  [Wu. et al, ECCV18], [Lu et al, CVPR19], [Cheng et al, CVPR19] [Rippel et al ICCV19], [Hu et al, 2020], [Agustsson et al. 2020], [Golinski et al. 2020] [Habiban et al.2019], [Lu et al. 2020], [Liu el.al.2020], [DVC, Lu et al. 2019] . . .

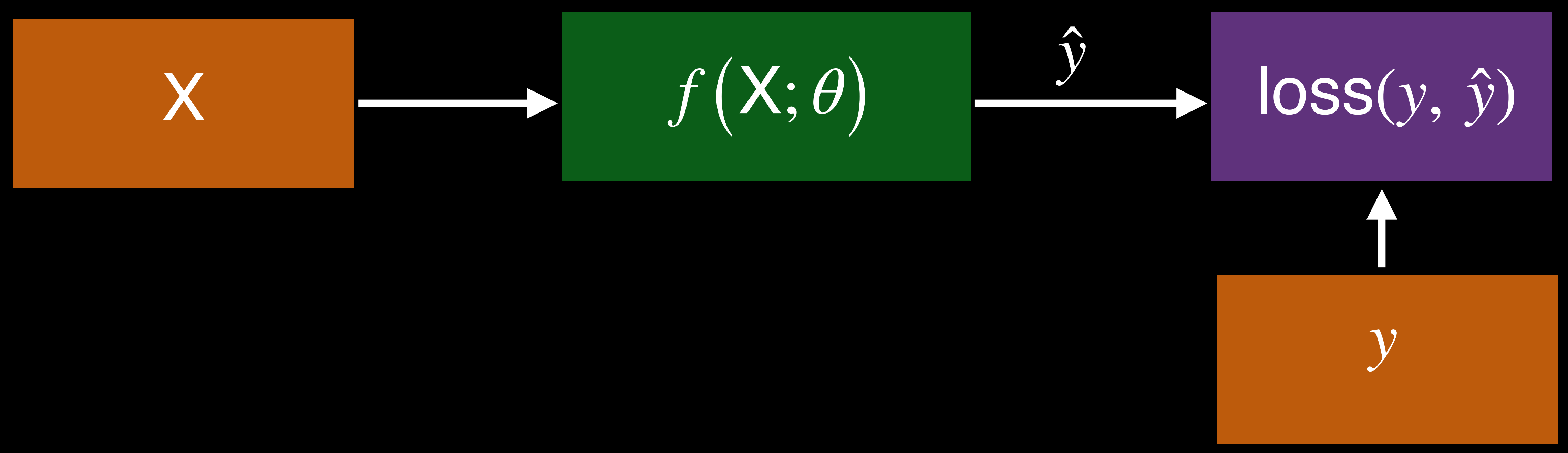
- **The CLIC Challenge**
  "Challenge on Learned Image Compression"
  -> ongoing image compression contest at Data Compression Conference

Lots of interesting works!

Our Goal: Understand the Key concepts

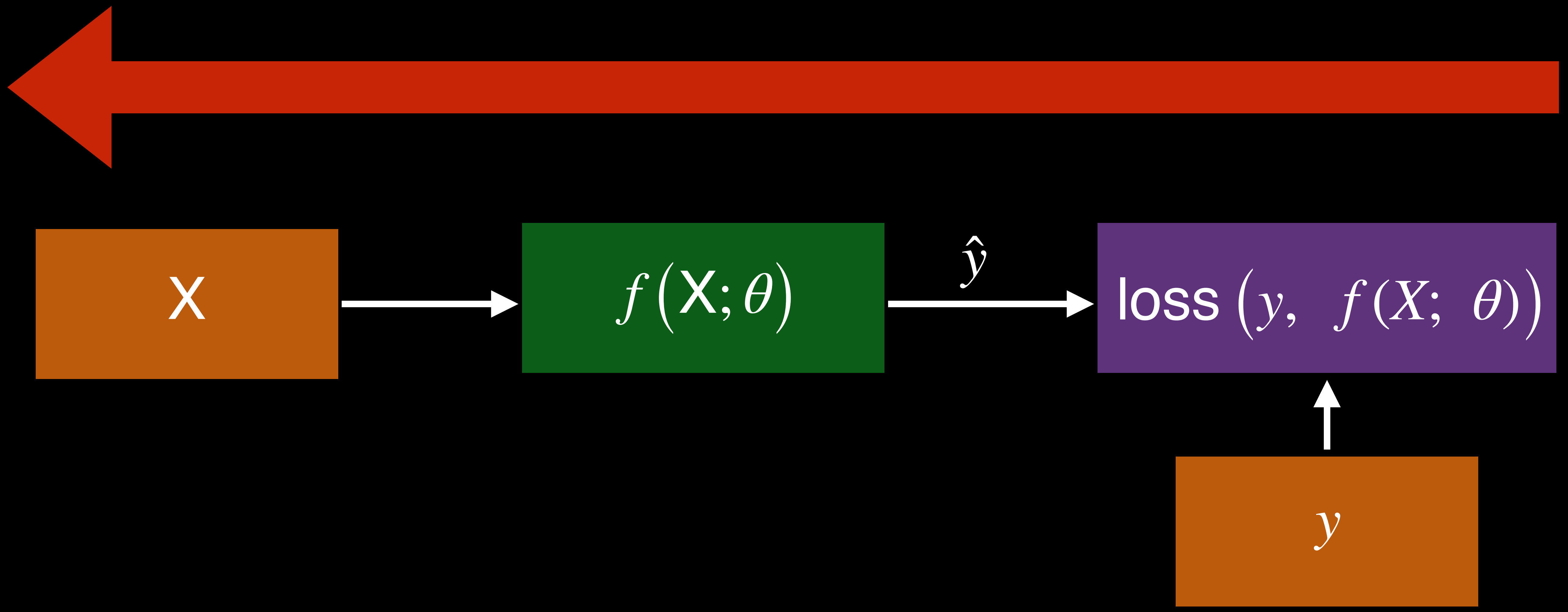
ML 101 Review

- **Data:** $(\mathbf{X},\ y)$
- **Non-linear Model and Differentiable Architectures:** $f(\mathbf{X};\theta)$
- **Loss Function:** loss$(y,\ \hat{y})$



ML 101 Review

- Use back-propagation/gradients to ***"learn"*** (update) model parameters θ
  - Does this by taking gradients (back-propagation) of loss $\left(y,\ f(X;\ \theta)\right)$ wrt θ

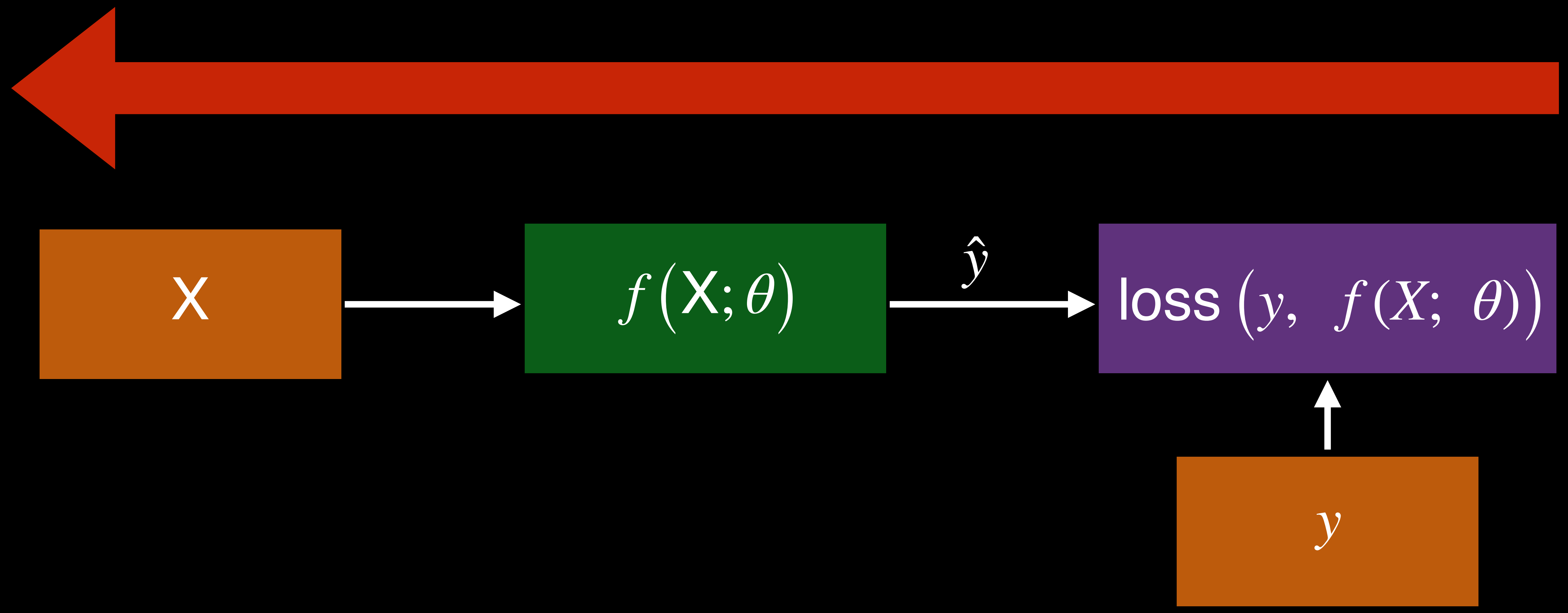


$$min_\theta \ \text{loss}\left(y,\ f(X;\ \theta)\right)$$

$$\theta_n = \theta_{n-1} - \nabla_\theta\Big(\text{loss}\left(y,\ f(X;\ \theta)\right)\Big)$$

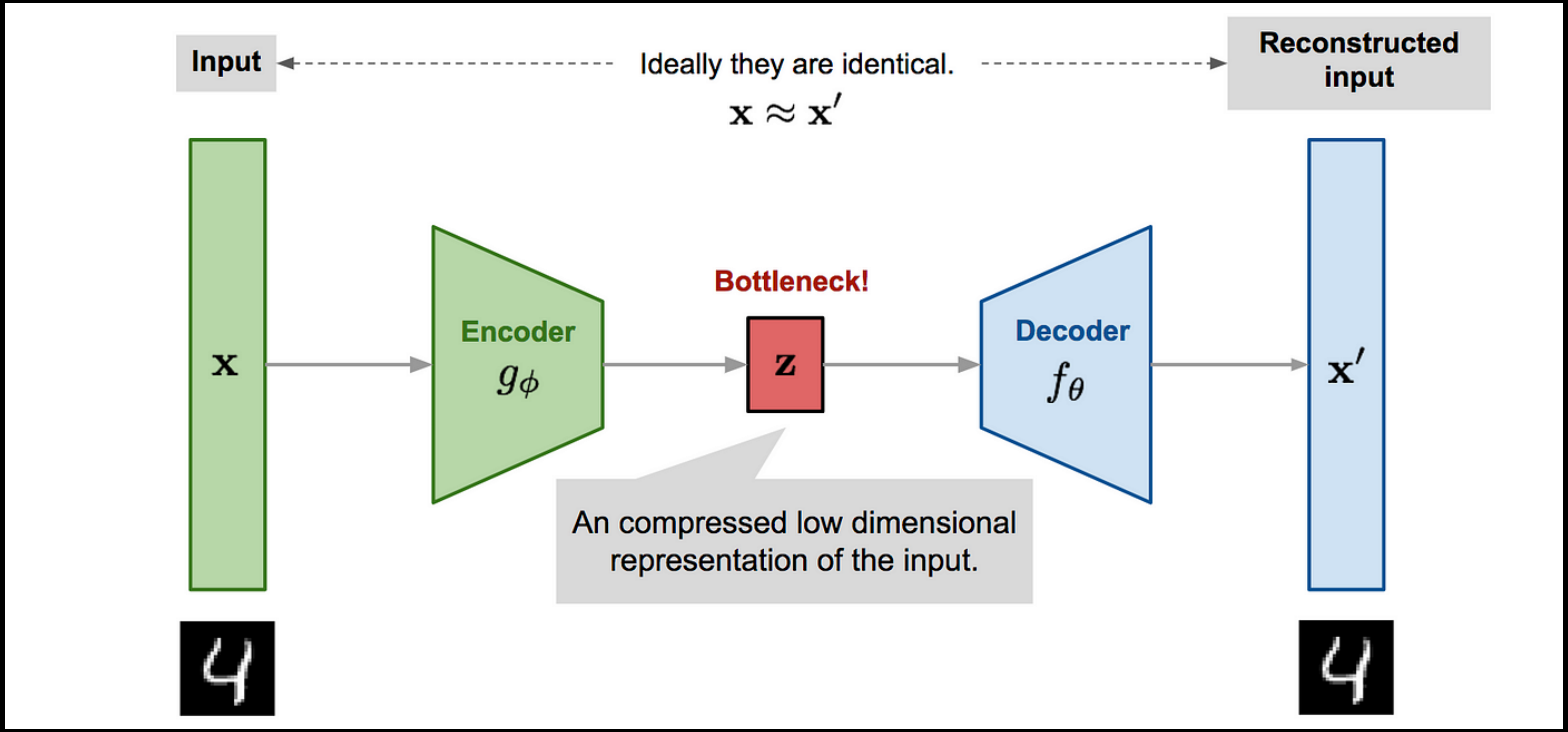
ML 101 Review

- Uses data to learn the model parameters θ for optimizing a $\text{loss}(\,\cdot\,)$
- Does this by taking gradients (back-propagation) of $\text{loss}\left(y,\ f(X;\ \theta)\right)$ wrt θ
- Advantage over standard models:
  - Allows optimizing any objective function loss as long as $f(\,\cdot\,;\ \theta)$ is differentiable!
    - doesn't play well with ***discrete*** distributions
  - Allows stacking of non-linear layers (linear layer + non-linearity)
    - recall stacking of linear layers is not so-powerful: ABC...= D if ABCD are matrices



For more details check out CS231N course notes: https://cs231n.github.io/

ML 101 Review: Auto-encoder architecture



https://lilianweng.github.io/posts/2018-08-12-vae/

The Image Compression Problem

I

Target Image

Encoder $\rightarrow$

$bits$

Decoder $\rightarrow$

$\hat{I}$

Target reconst

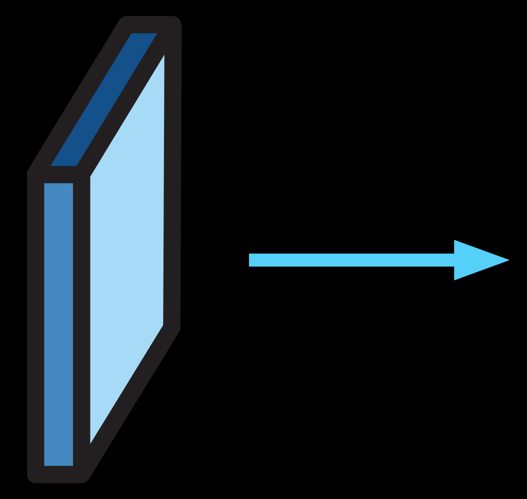
Goal: $$\min_{L(bits)\leq B} d(I, \hat{I})$$

Rate

Distortion

Traditional Image Codecs

I

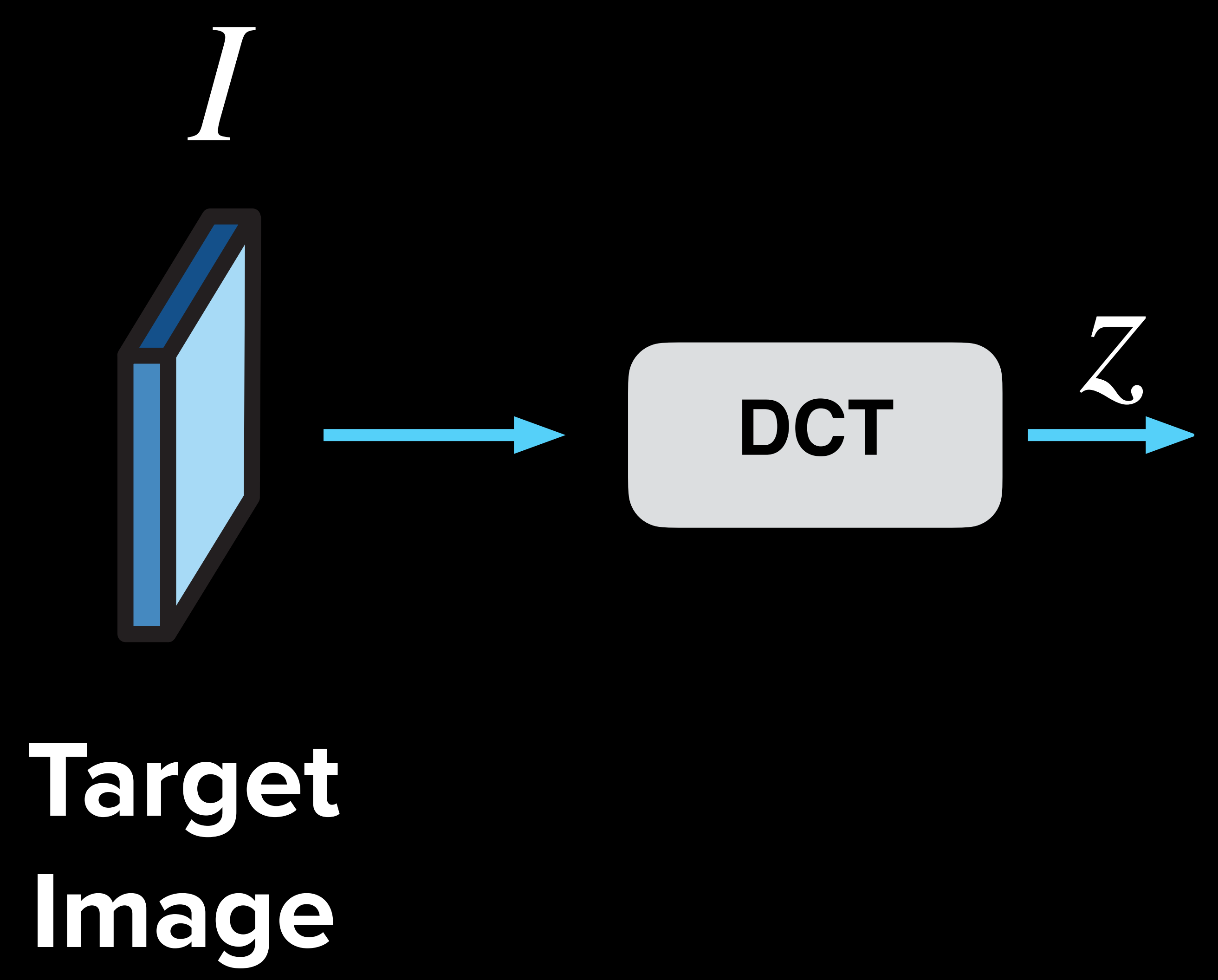


Target Image

Encoding proceeds in 3 steps:

CAUTION: Simplified framework

Traditional Image Codecs

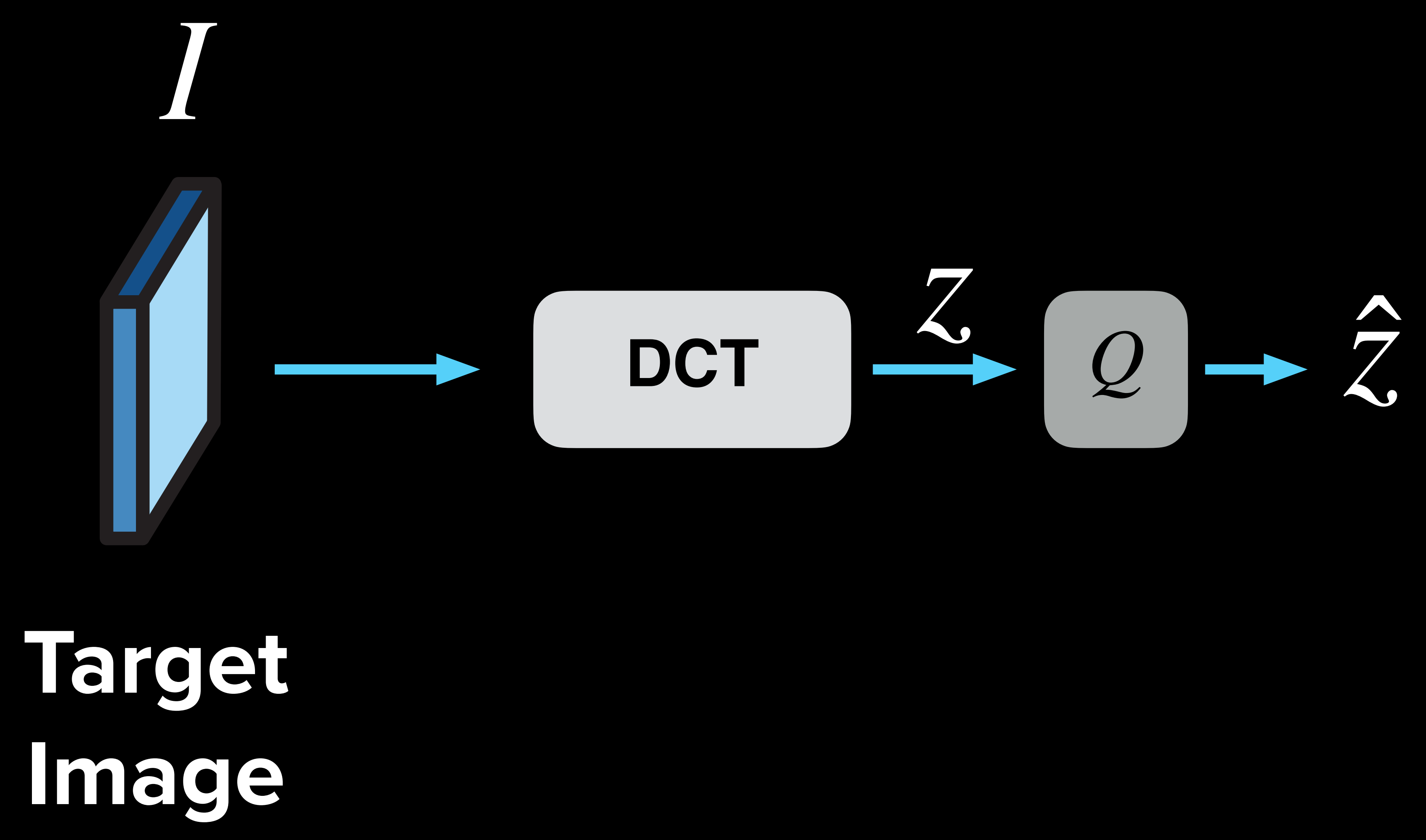


Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels

CAUTION: Simplified framework

Traditional Image Codecs

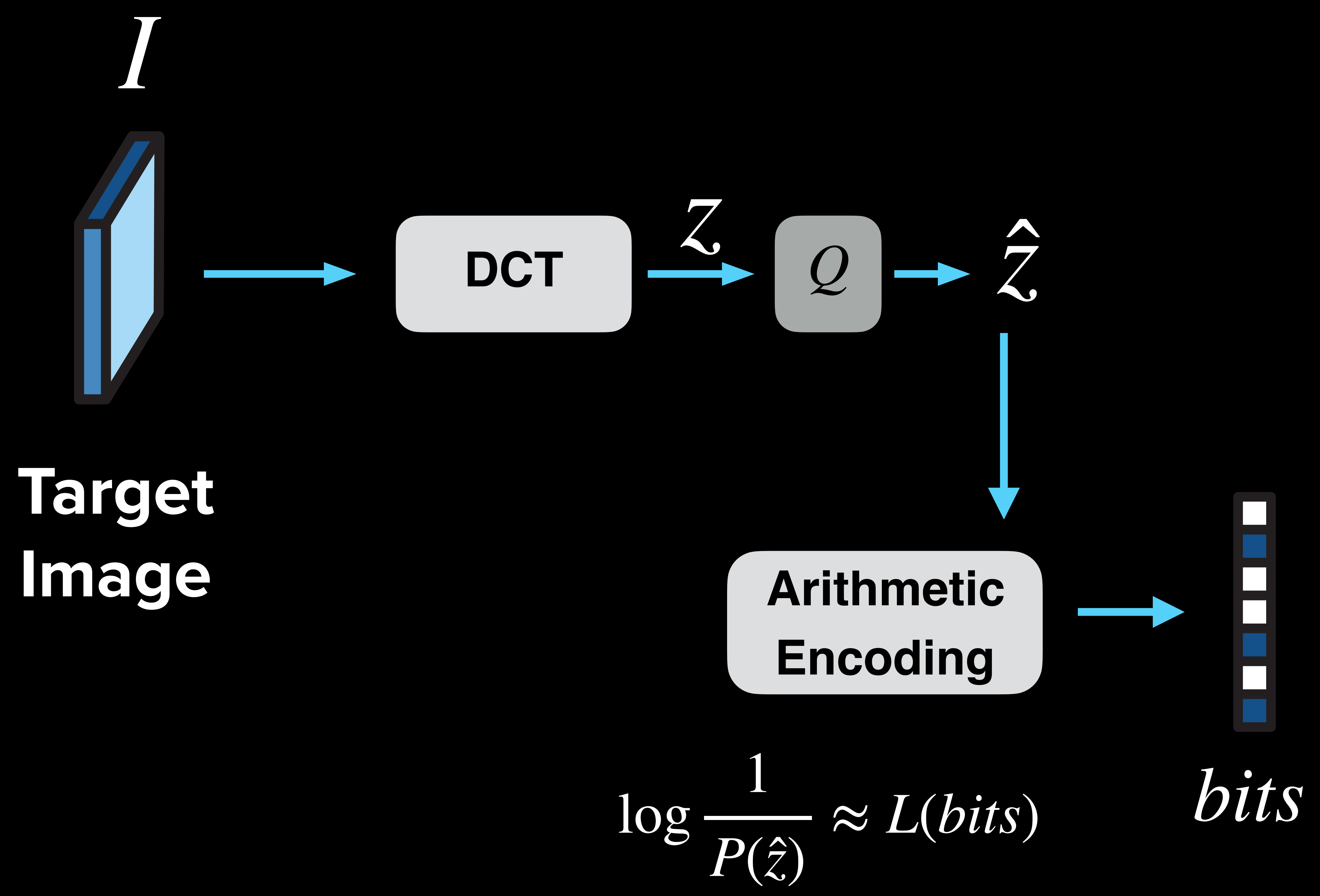


Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels
- **Quantize ->** Loss of precision
  $Q([2.3, 3.7]) = [2,4]$

CAUTION: Simplified framework

Traditional Image Codecs



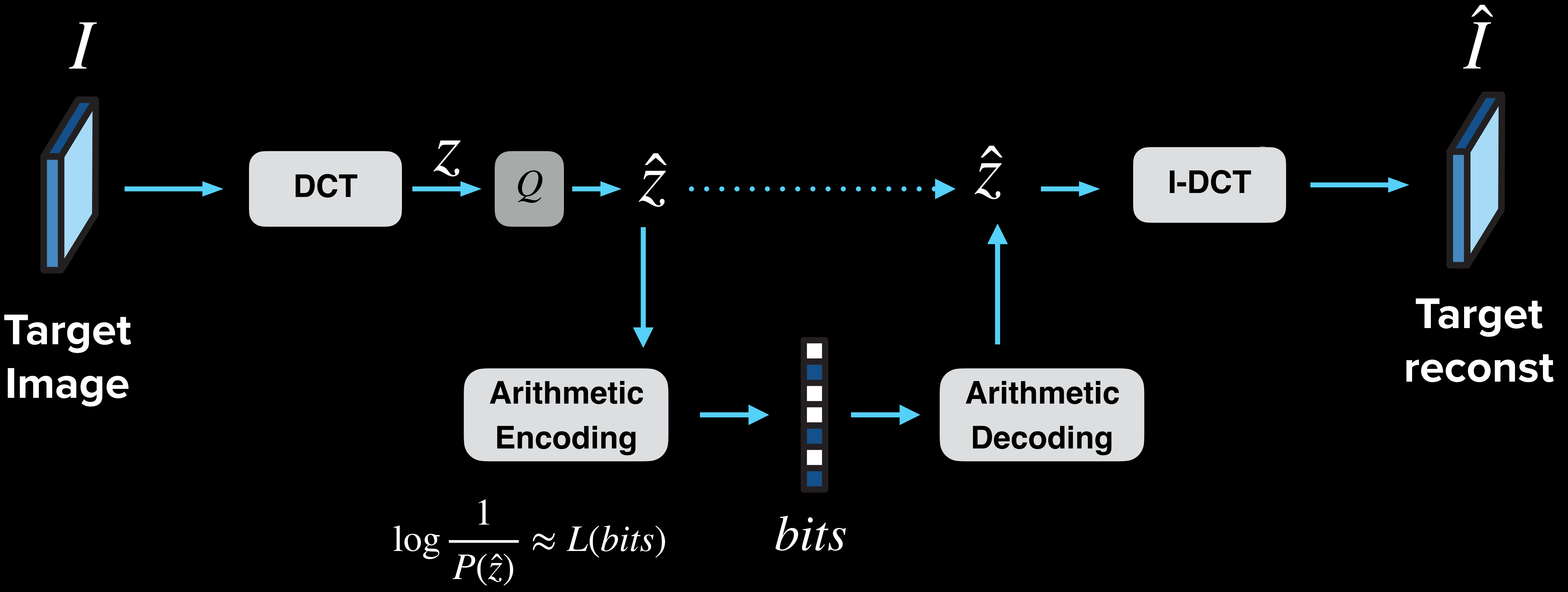
Encoding proceeds in 3 steps:

- **DCT Transform:**
  Linear transform to decorrelate the pixels
- **Quantize ->** Loss of precision
  $Q([2.3, 3.7]) = [2,4]$
- **Arithmetic/Huffman ->** Lossless Compression
  In the simplest form, uses a discrete distribution $P(\hat{z})$ to encode $\hat{z}$ into a bitstream of length

$$L(bits) \approx \log \frac{1}{P(\hat{z})}$$

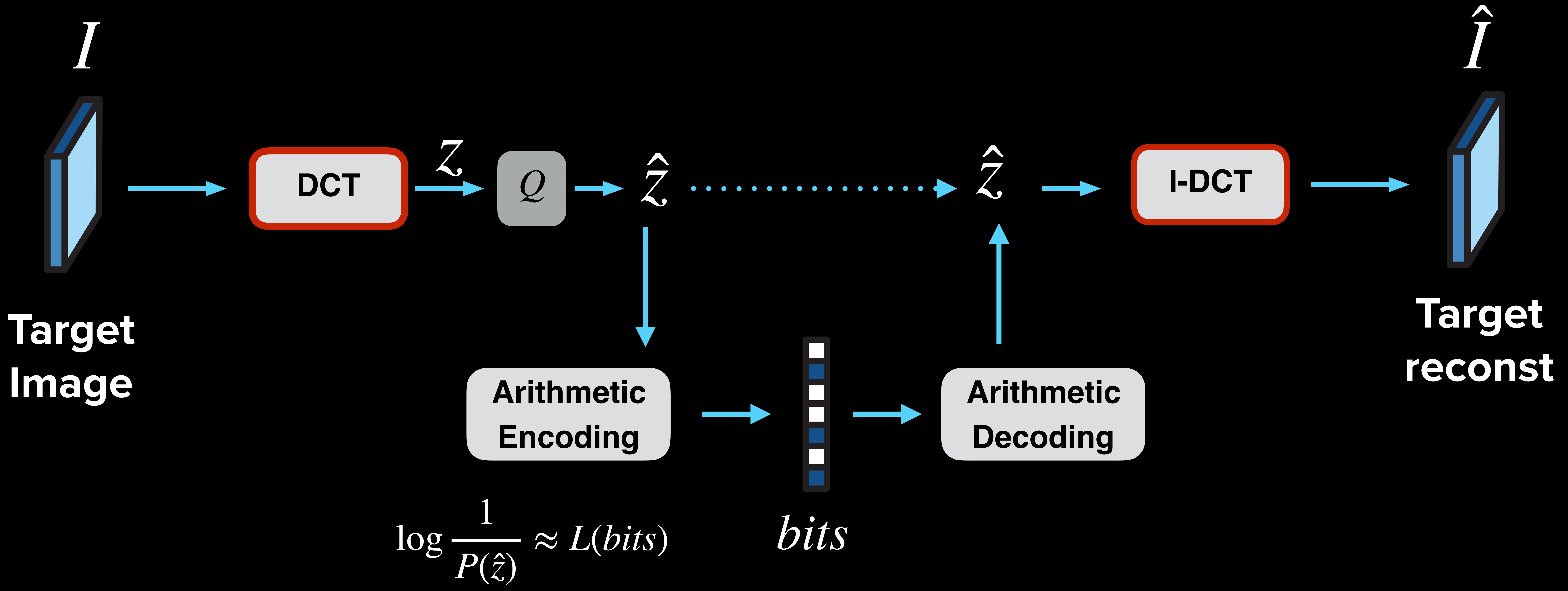
CAUTION: Simplified framework

Traditional Image Codecs



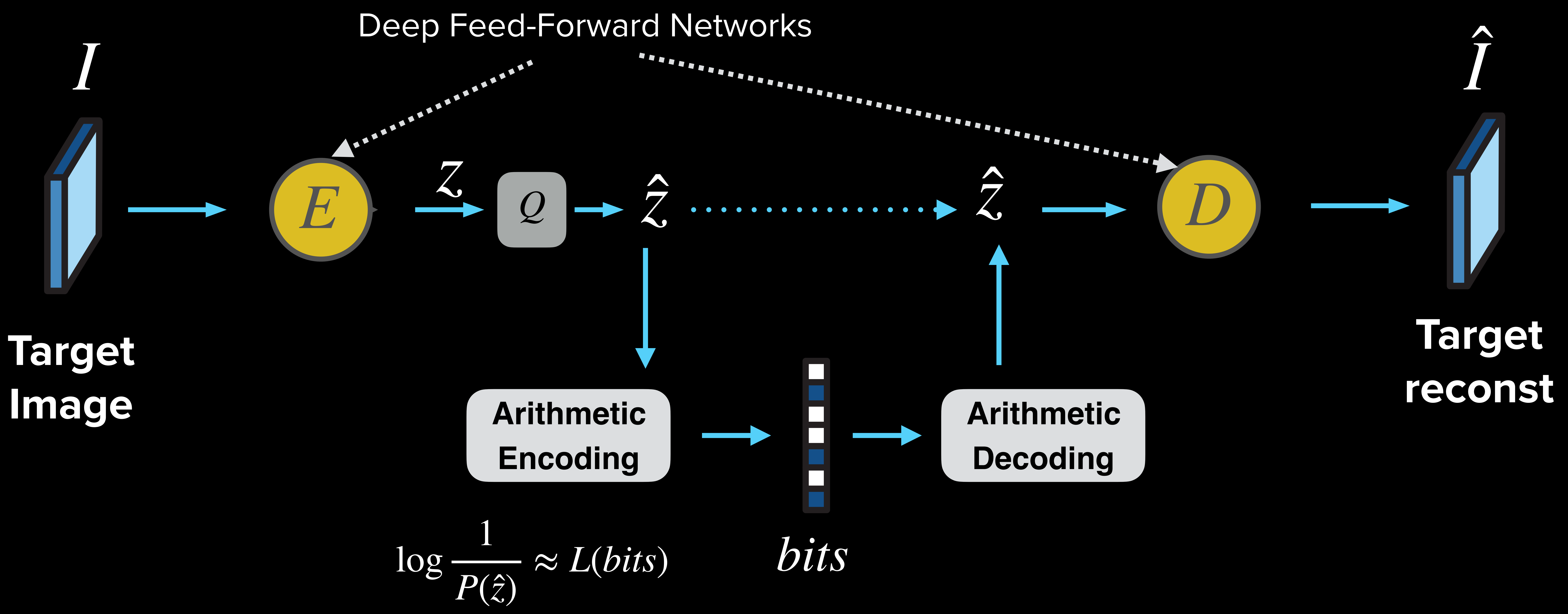
Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Traditional Image Codecs

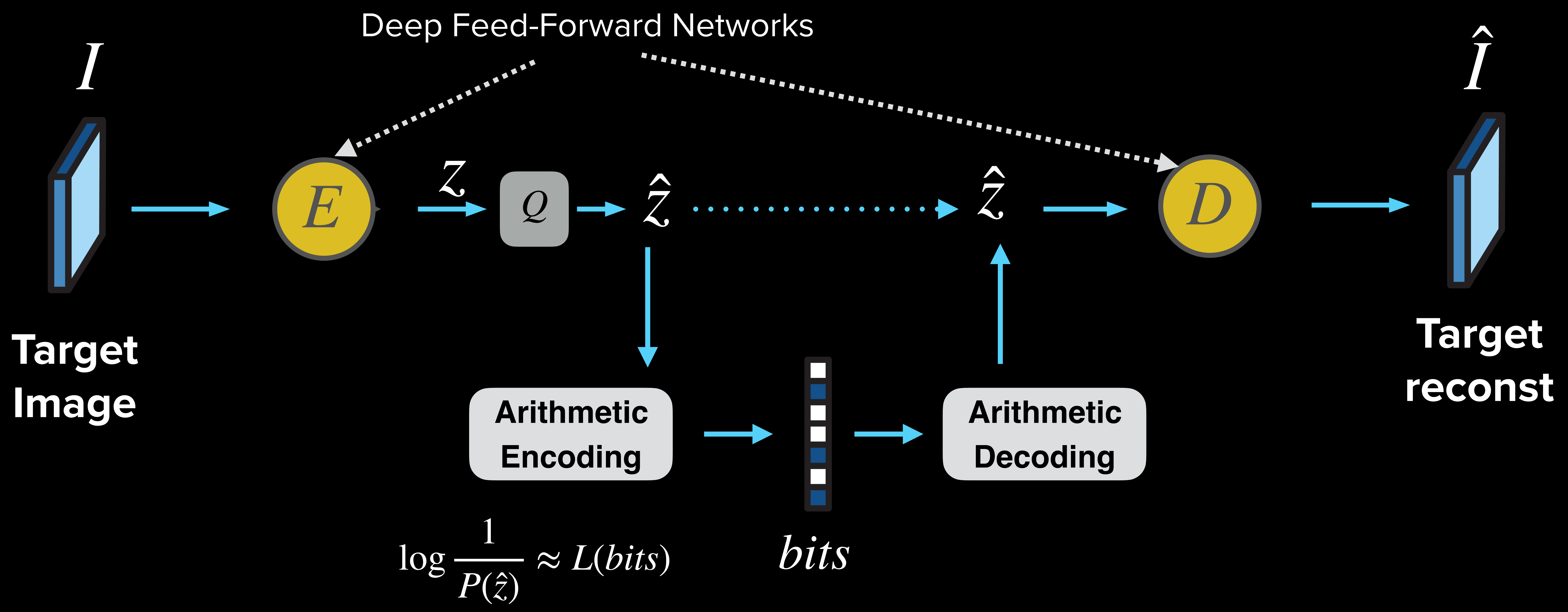


Goal: $\min_{L(bits) \leq B} d(I, \hat{I})$

Learned Image Codecs

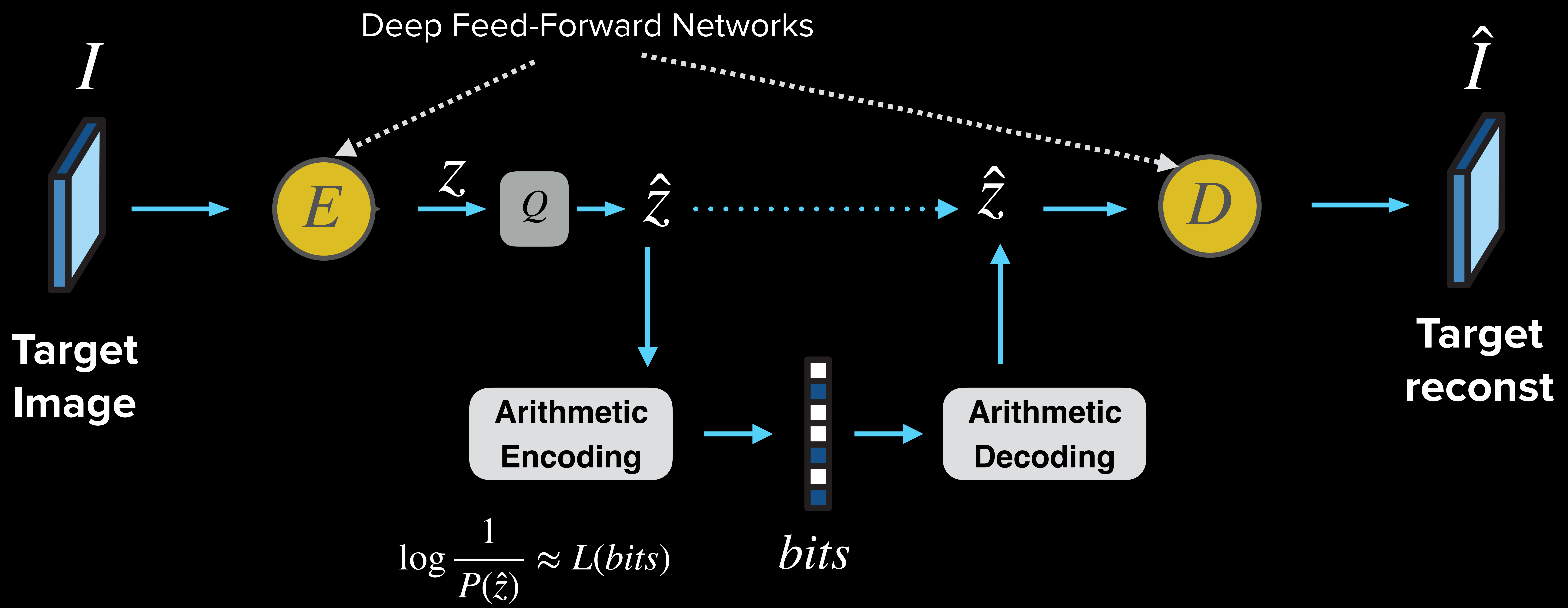


Learned Image Codecs



Question: How do we train the parameters?

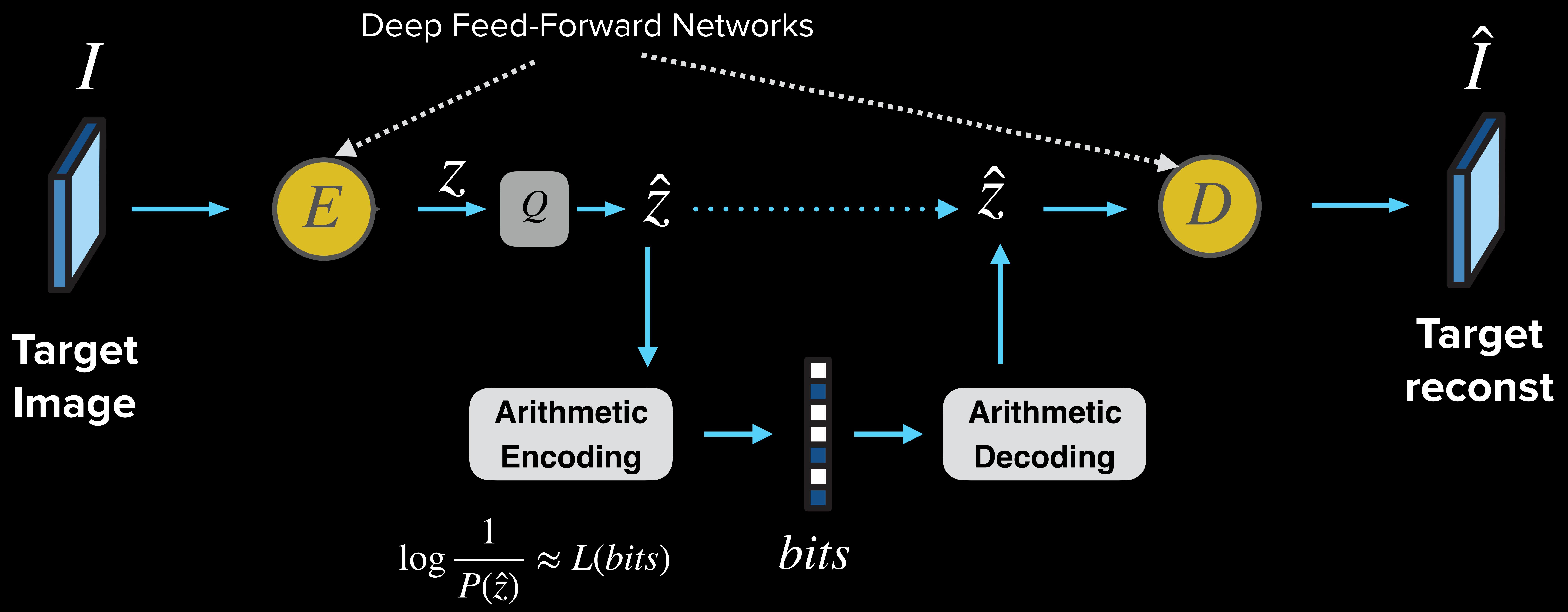
Learned Image Codecs



Loss Function = $L(bits) + \lambda d(I, \hat{I})$

Rate

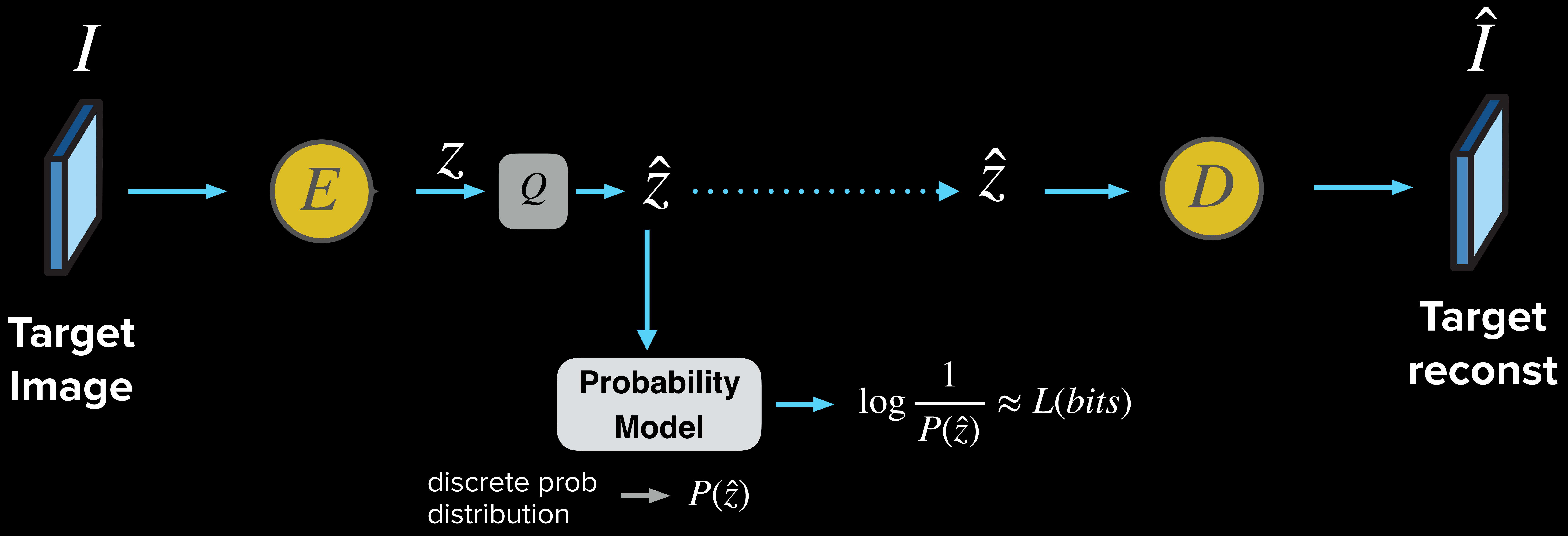
Learned Image Codecs



Loss Function = $L(bits) + \lambda d(I, \hat{I}) \approx \log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

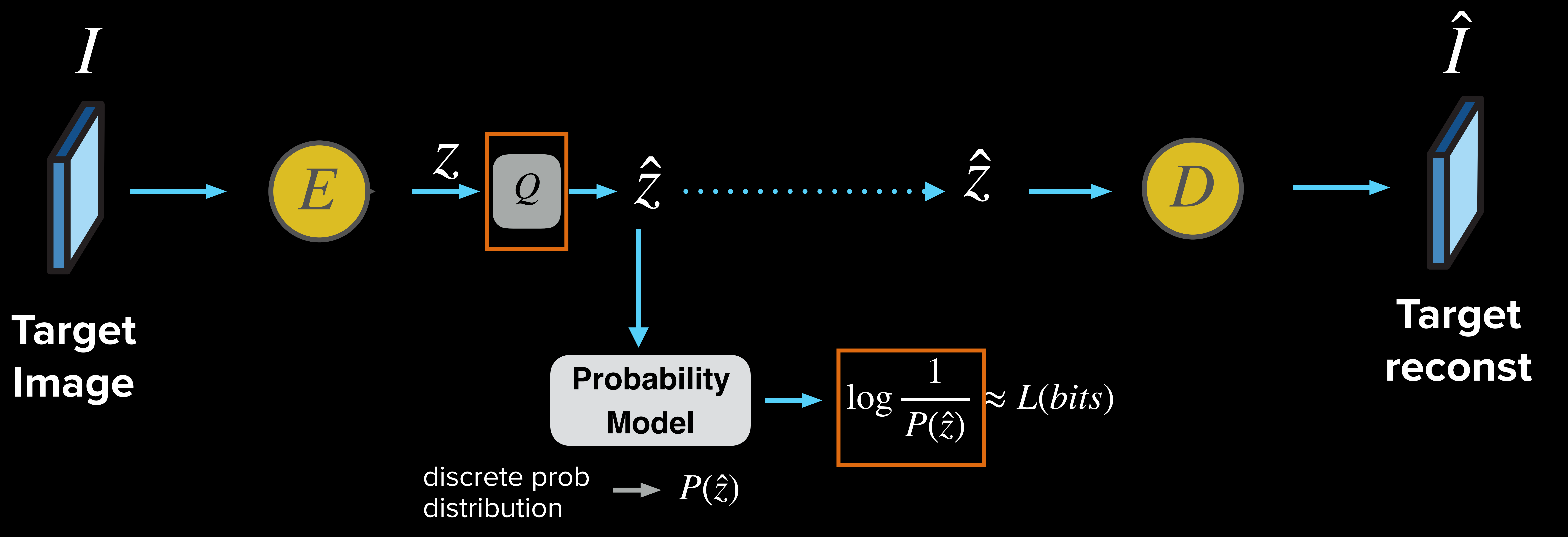
Rate

Learned Image Codecs



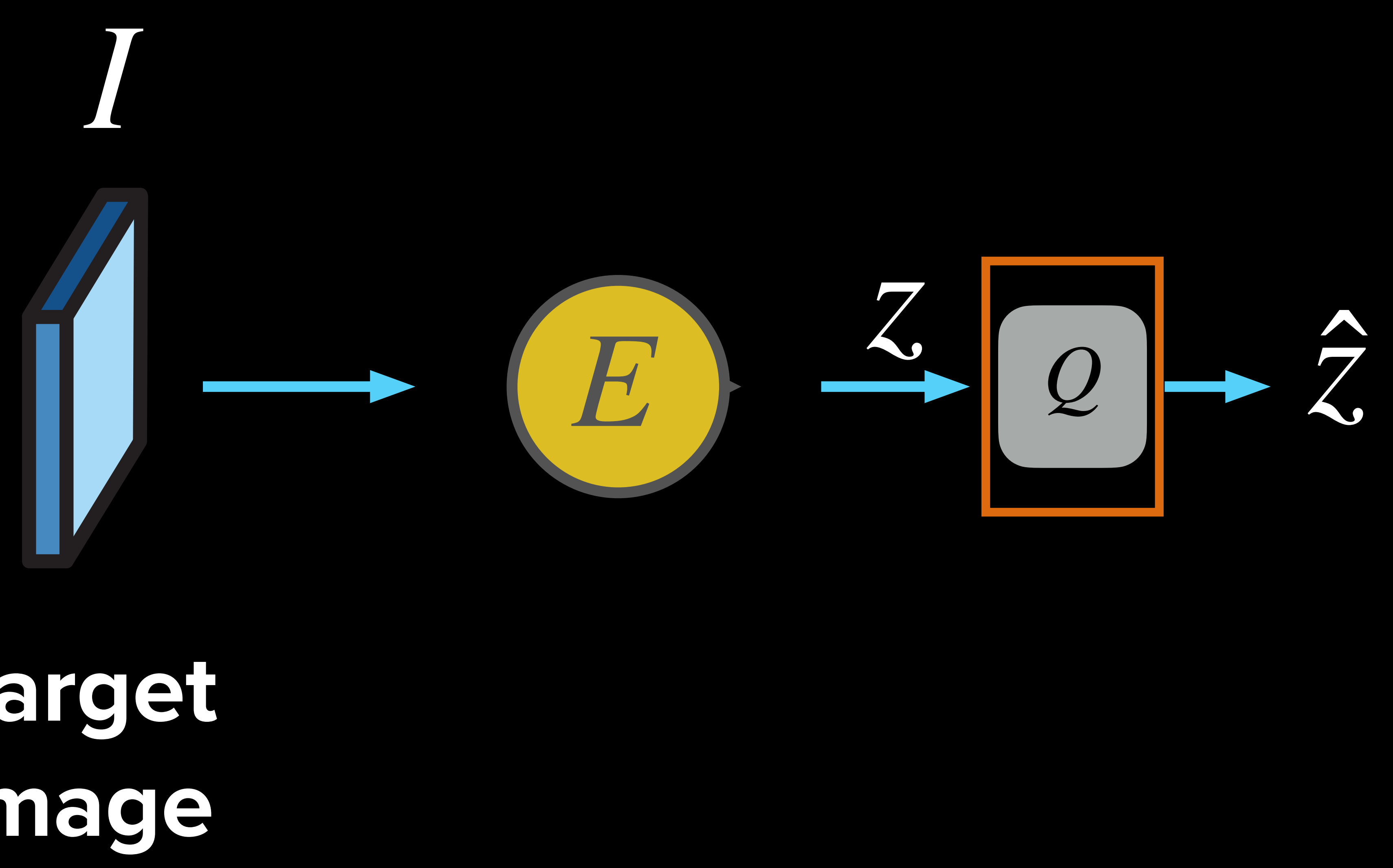
Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

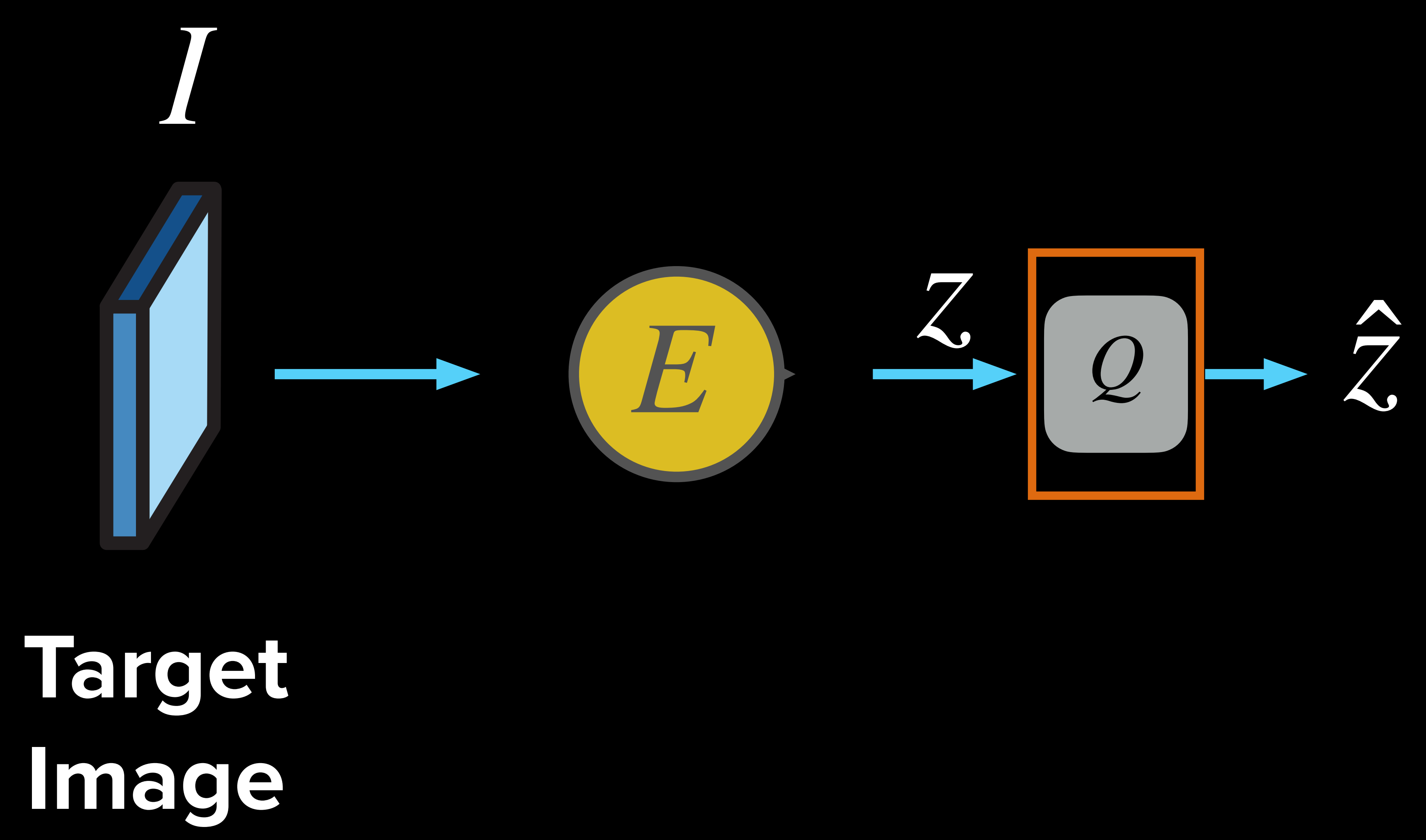
Learned Image Codecs



- **Quantizer** -> $Q([2.3, 3.7]) = [2,4]$

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

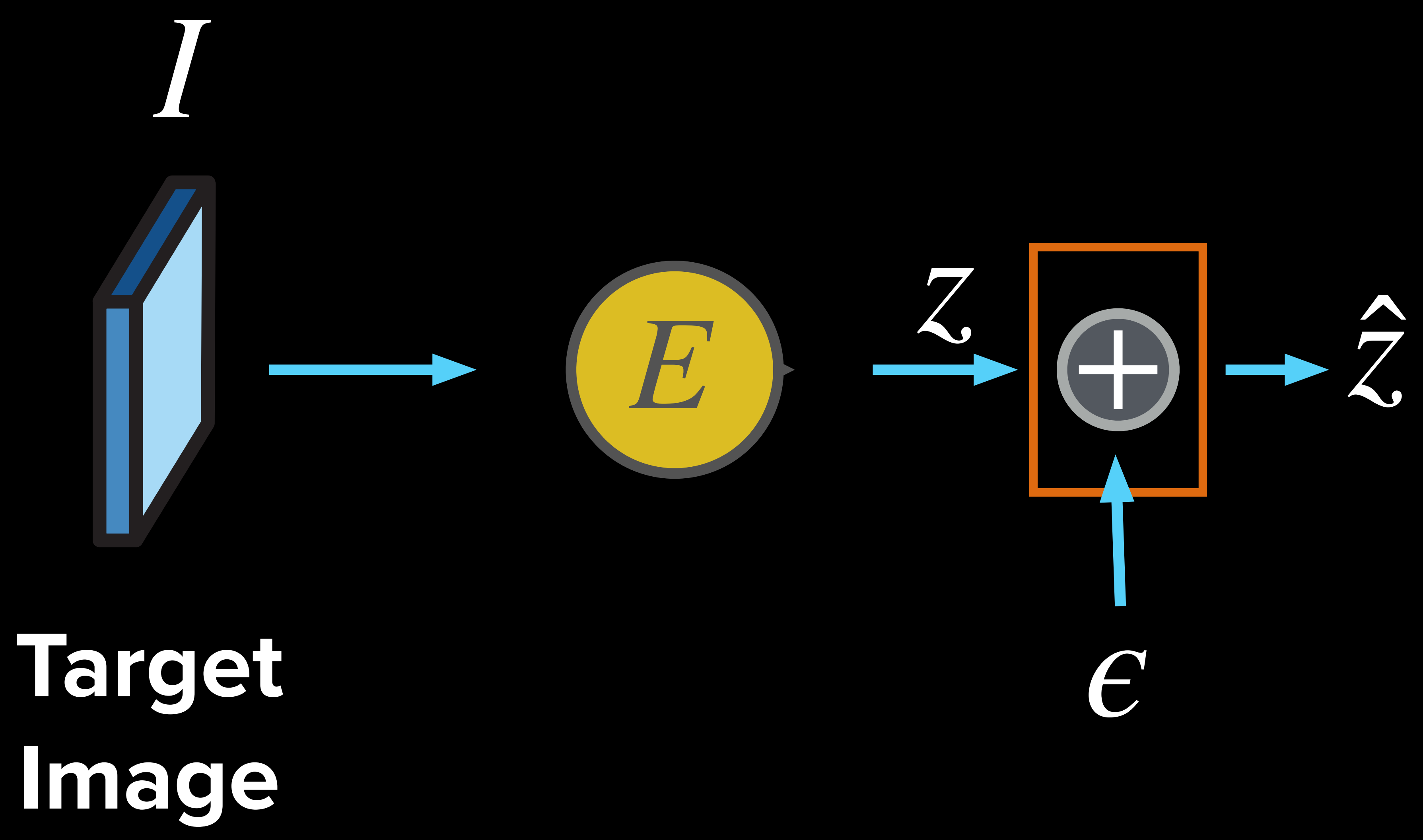
Learned Image Codecs



- **Quantizer** -> $Q([2.3, 3.7]) = [2, 4]$
- **Workaround-1:** model the quantizer as adding noise during training $\hat{z} = z + \epsilon$, where $\epsilon \sim U(-0.5, 0.5)$

Loss Function = $\log \dfrac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

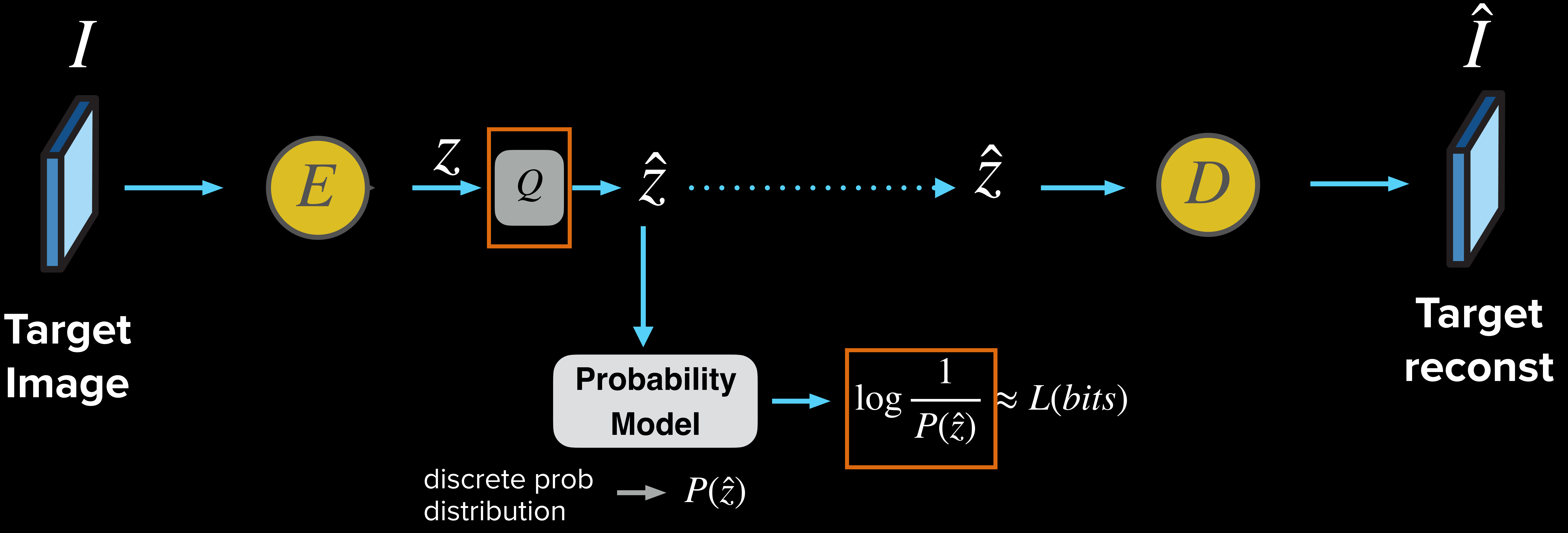
Learned Image Codecs



- **Quantizer** -> $Q([2.3,3.7]) = [2,4]$
- **Workaround-1:** model the quantizer as adding noise during training $\hat{z} = z + \epsilon$, where $\epsilon \sim U(-0.5,0.5)$

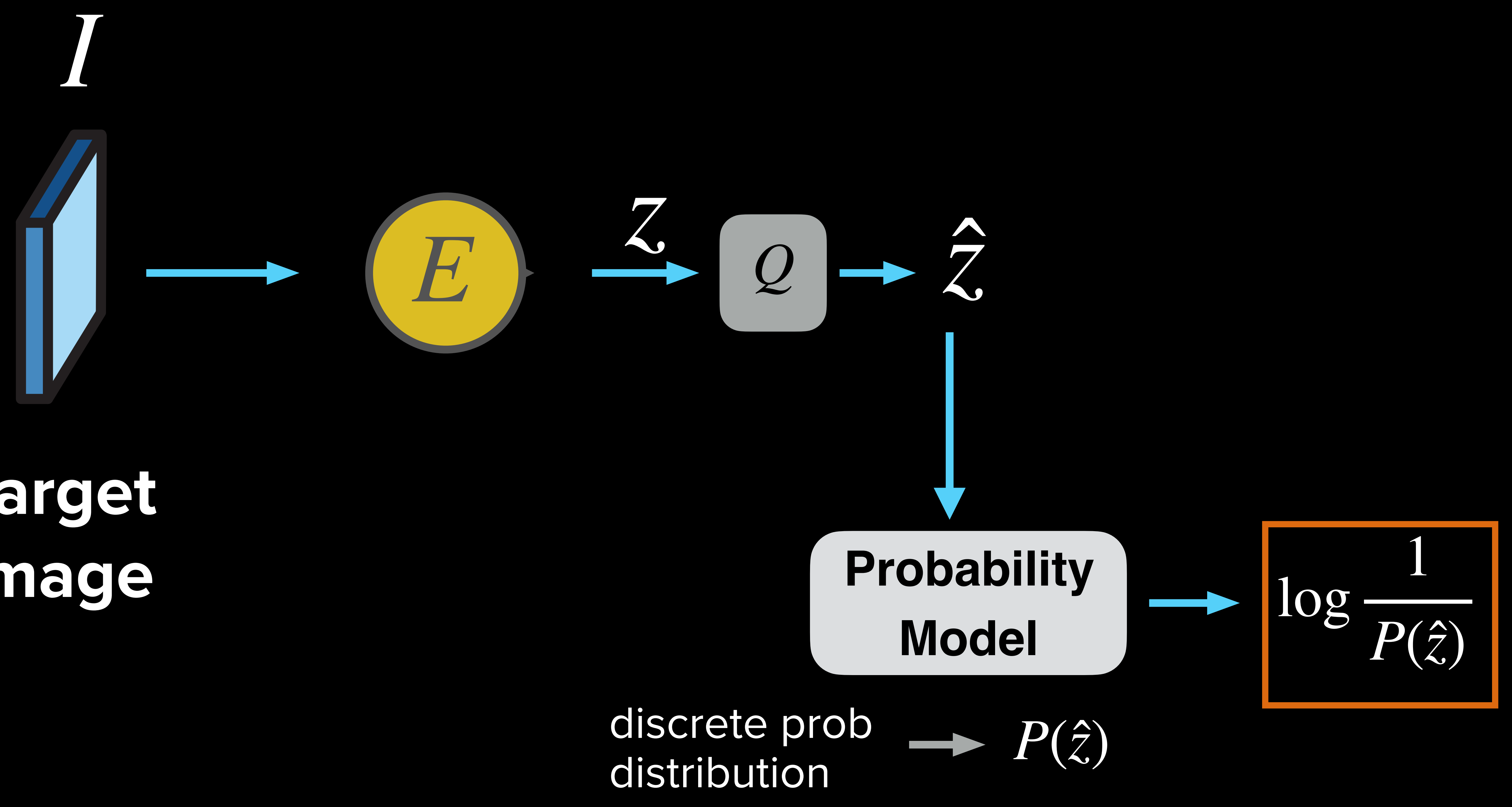
Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

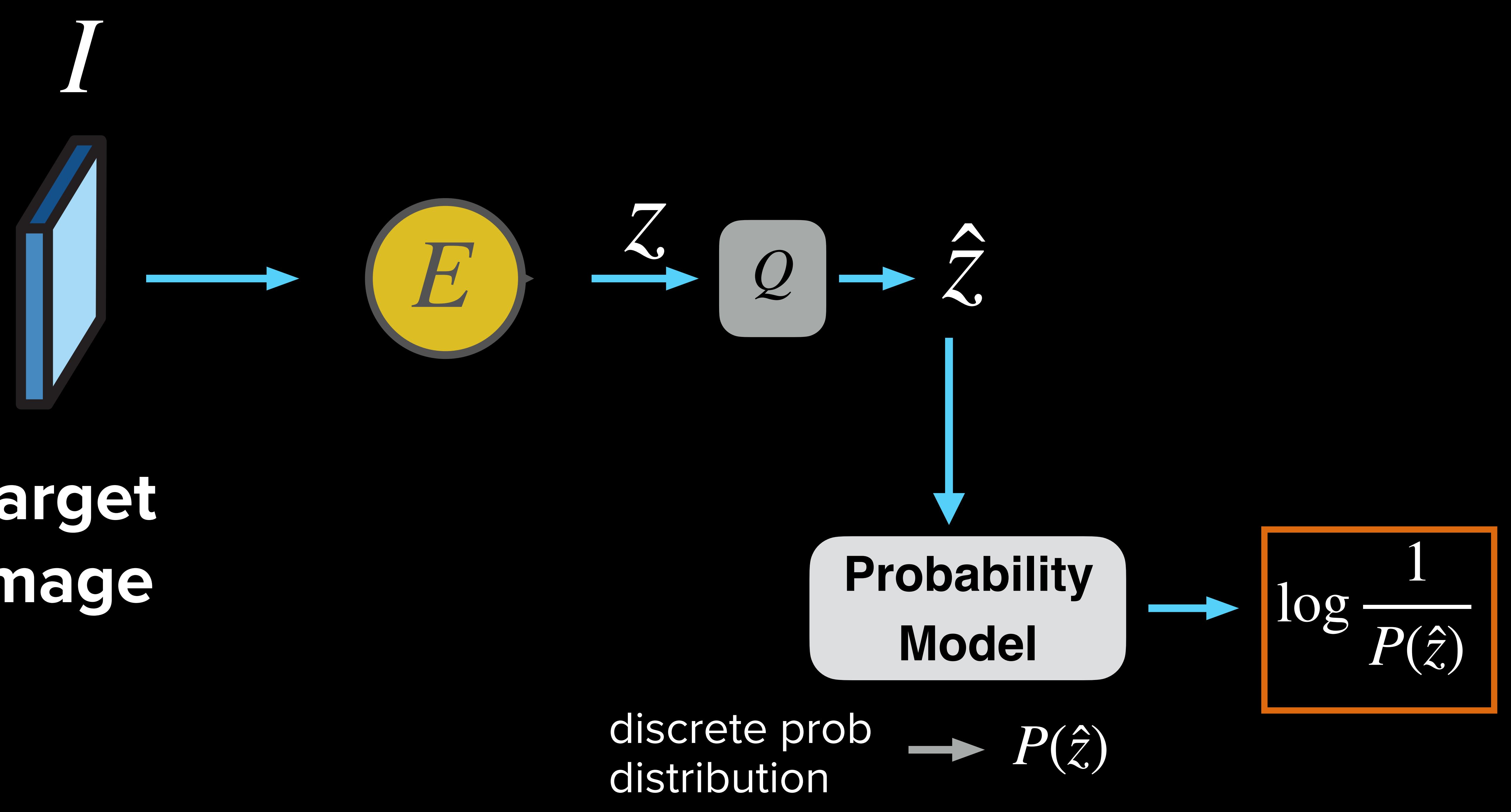
Learned Image Codecs



- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

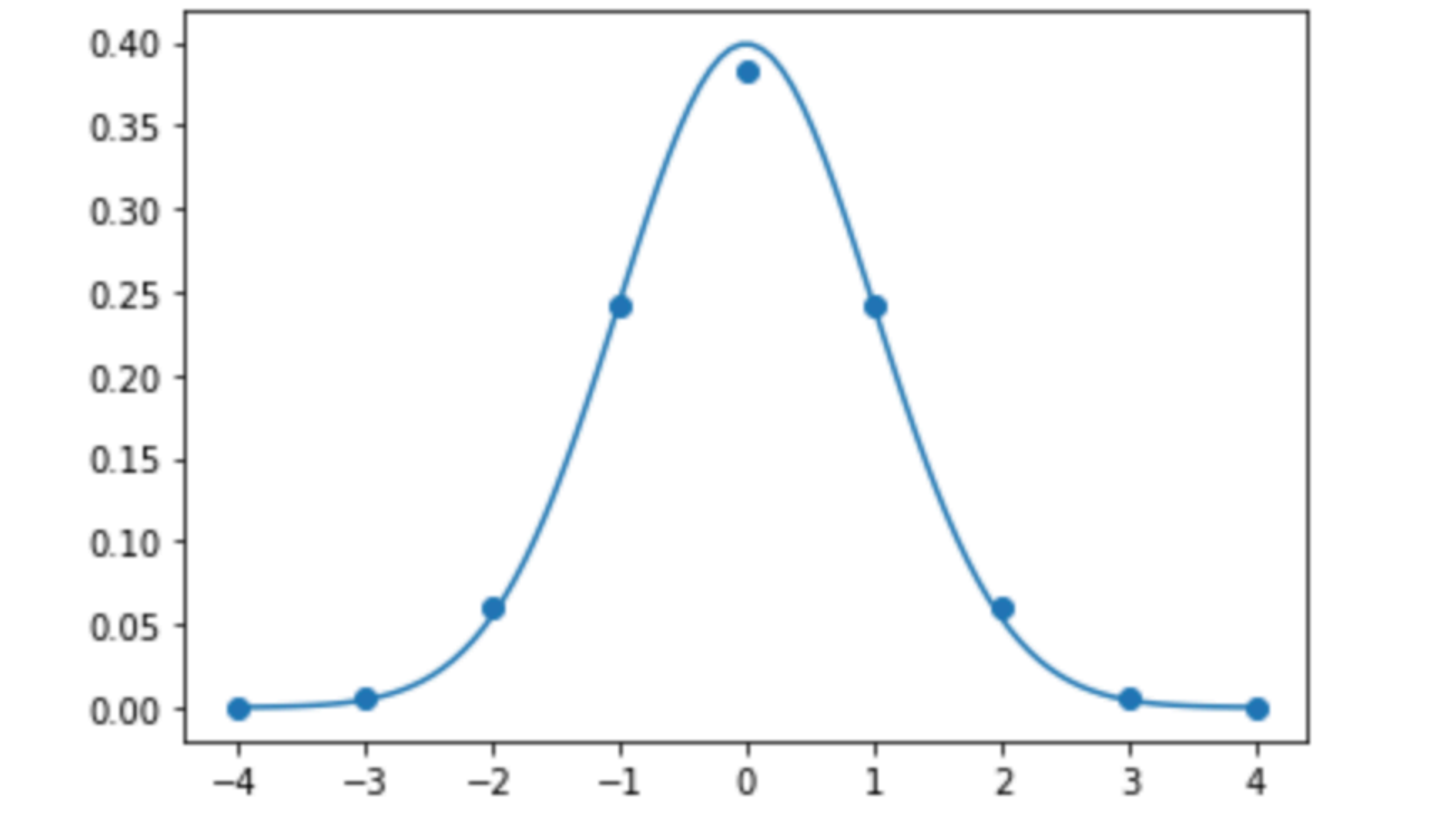
Learned Image Codecs



- $\dfrac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)
  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$

Loss Function = $\log \dfrac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Learned Image Codecs



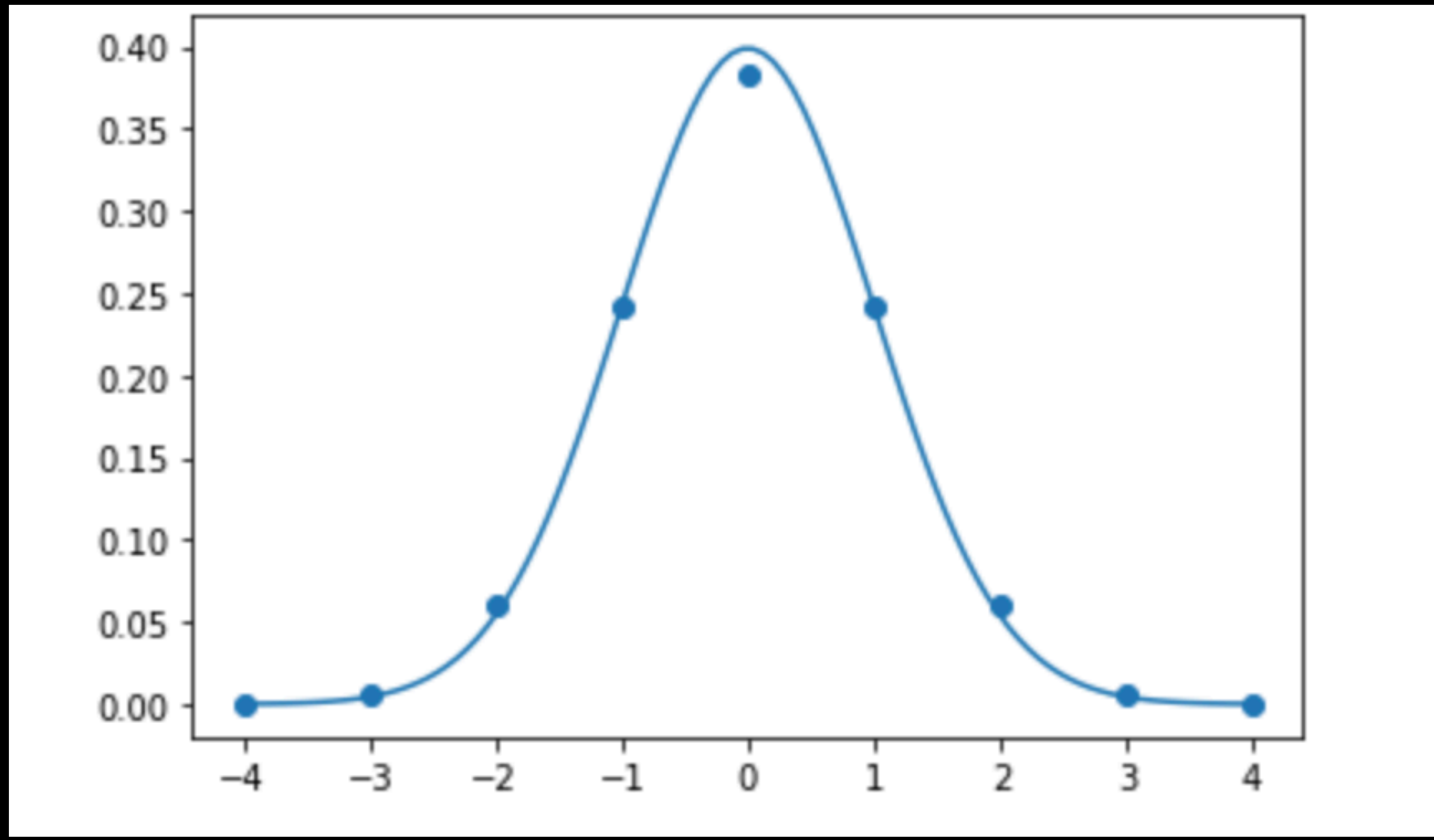
$P(\hat{z}) = GaussCDF(\hat{z} + 0.5) - GaussCDF(\hat{z} - 0.5)$

- $\dfrac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!

- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)

  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$

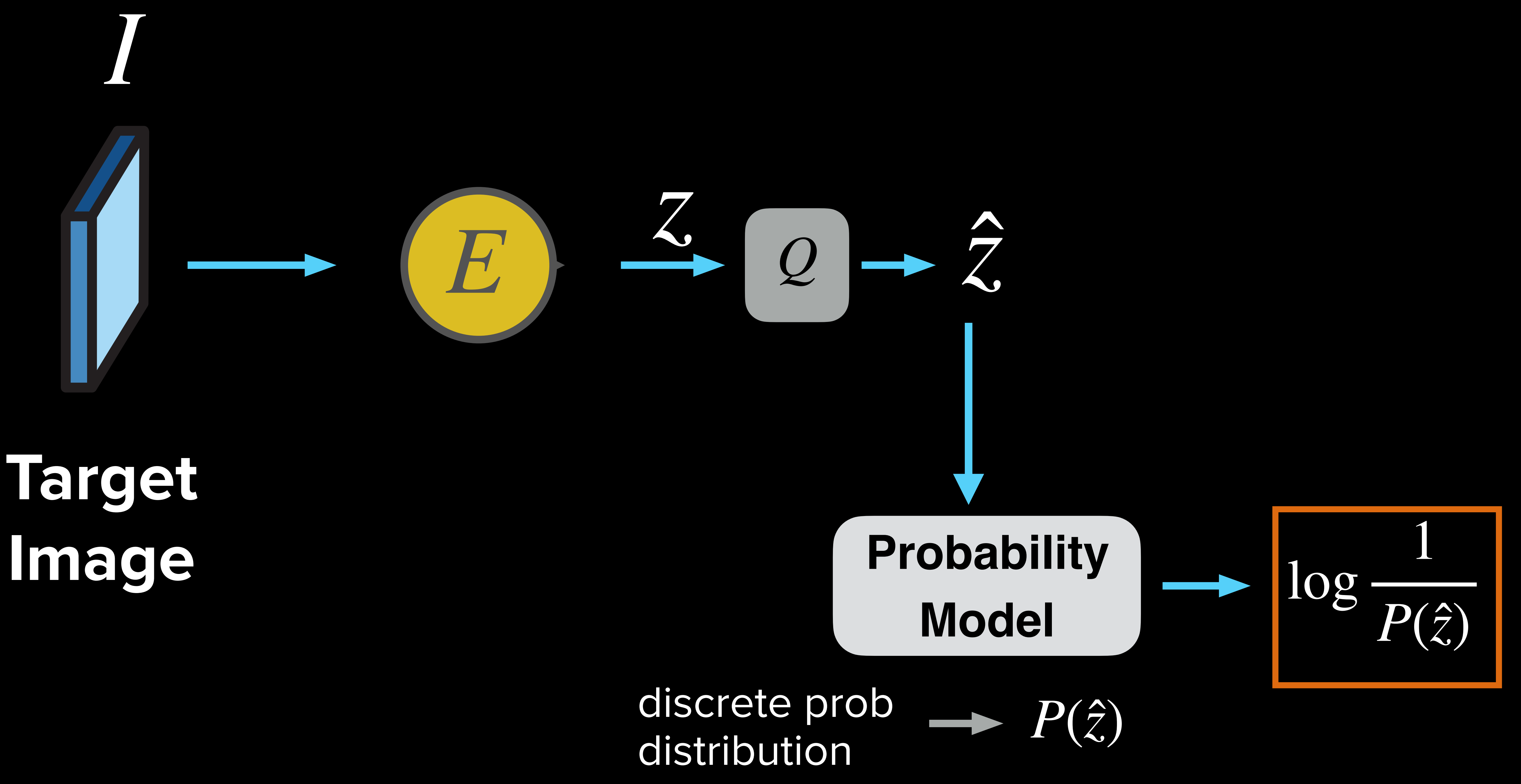
Learned Image Codecs



$P(\hat{z}) = GaussCDF(\hat{z} + 0.5) - GaussCDF(\hat{z} - 0.5)$

- $\dfrac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)
  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$
- Gradient is now well defined!
  $\dfrac{\partial P(\hat{z})}{\partial \hat{z}} = PDF(\hat{z} + 0.5) - PDF(\hat{z} - 0.5)$

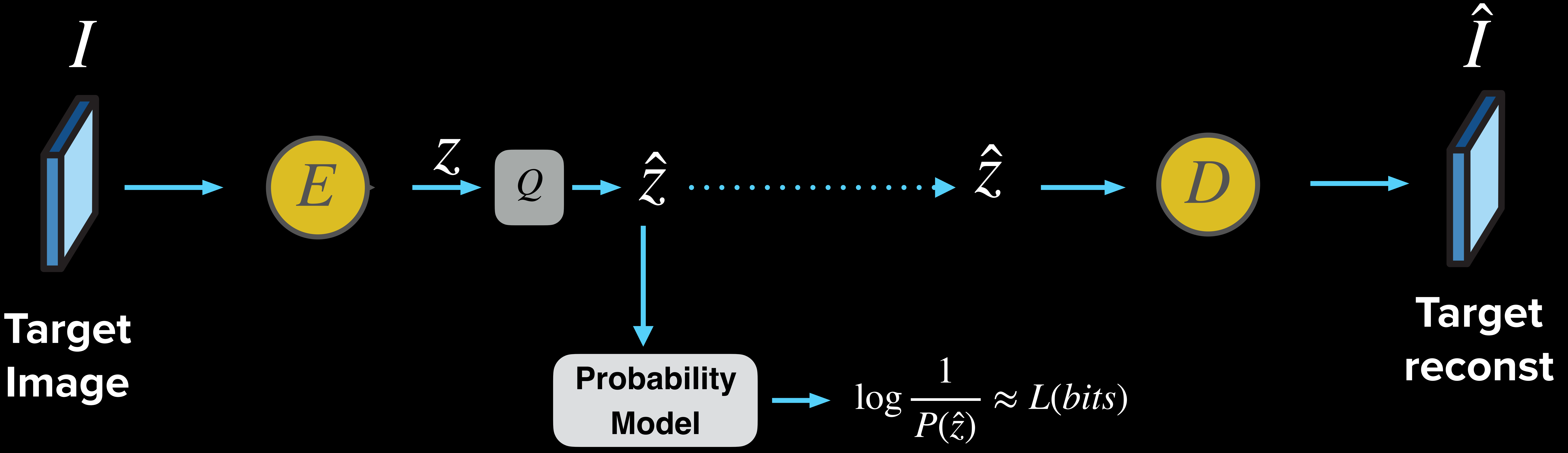
Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

- $\frac{\partial P(\hat{z})}{\partial \hat{z}}$ is not defined!
- **Idea:** Parametrize $P(\hat{z})$ using a density function (for ex: $\mathcal{N}(0,1)$)
  $P(\hat{z}) = CDF(\hat{z} + 0.5) - CDF(\hat{z} - 0.5)$
- Gradient is now well defined!
  $\frac{\partial P(\hat{z})}{\partial \hat{z}} = PDF(\hat{z} + 0.5) - PDF(\hat{z} - 0.5)$

End-to-End Learned Image Codec



$$P(\hat{z}) = GaussCDF(\hat{z} + 0.5) - GaussCDF(\hat{z} - 0.5)$$

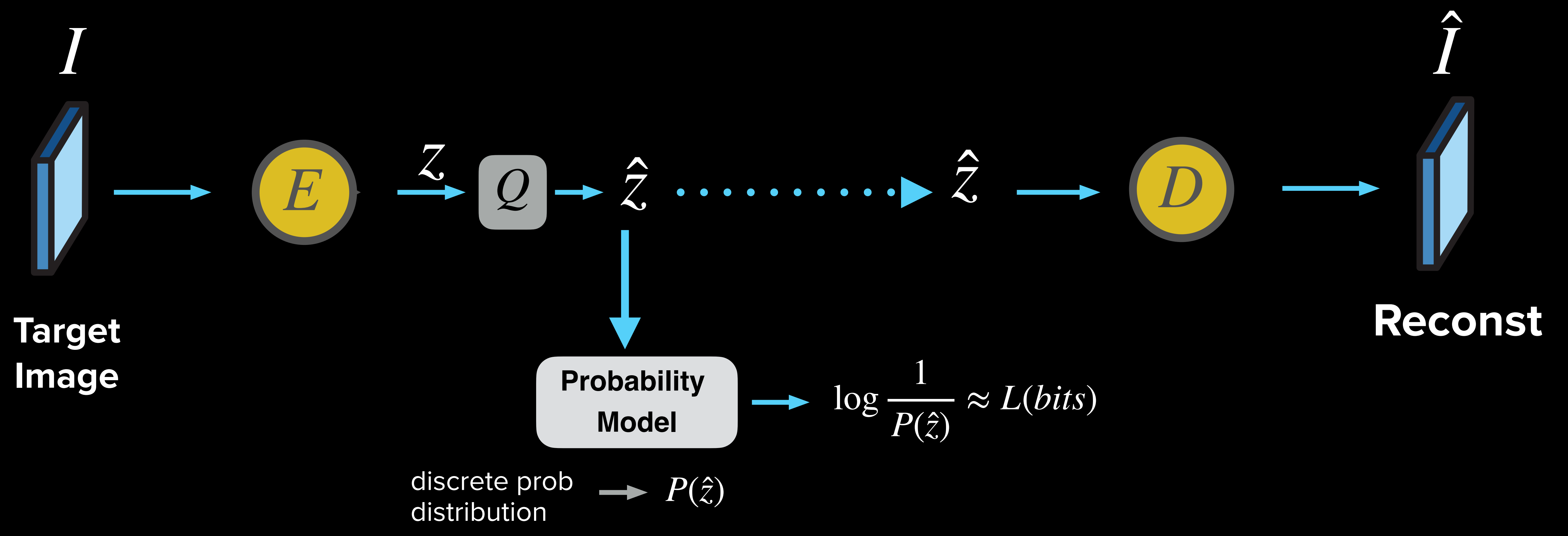
Possible to train end-to-end!

Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Example -> MNIST

https://colab.research.google.com/drive/1O3eQAaxlyLYI1HO7K1b12eJQsQKxjWwx?usp=sharing

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Possible to train end-to-end!

No hand-tuning parameters

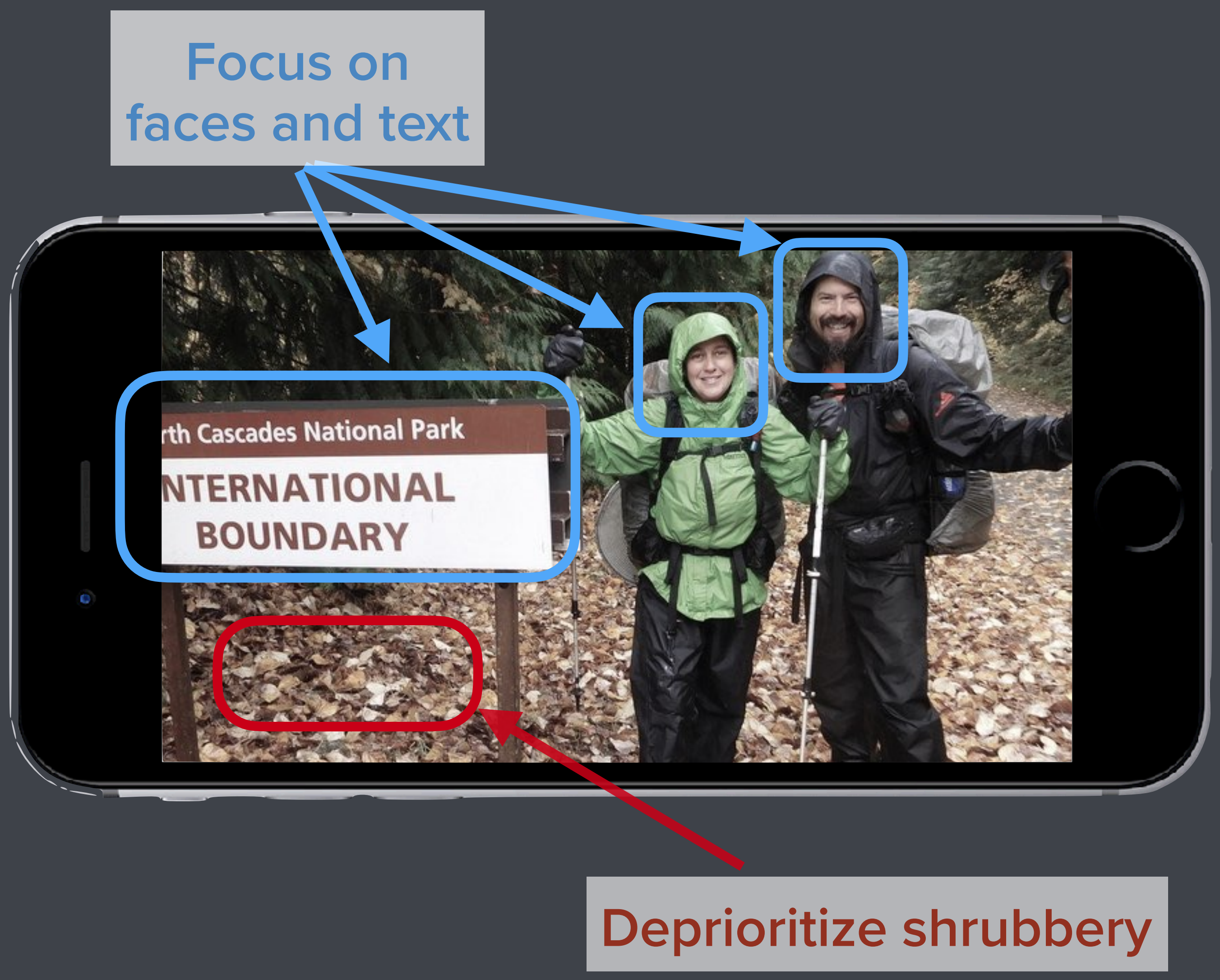
Can “learn” the parameters

ML Offers Adaptivity Not Possible With Traditional Codecs

Domain-aware

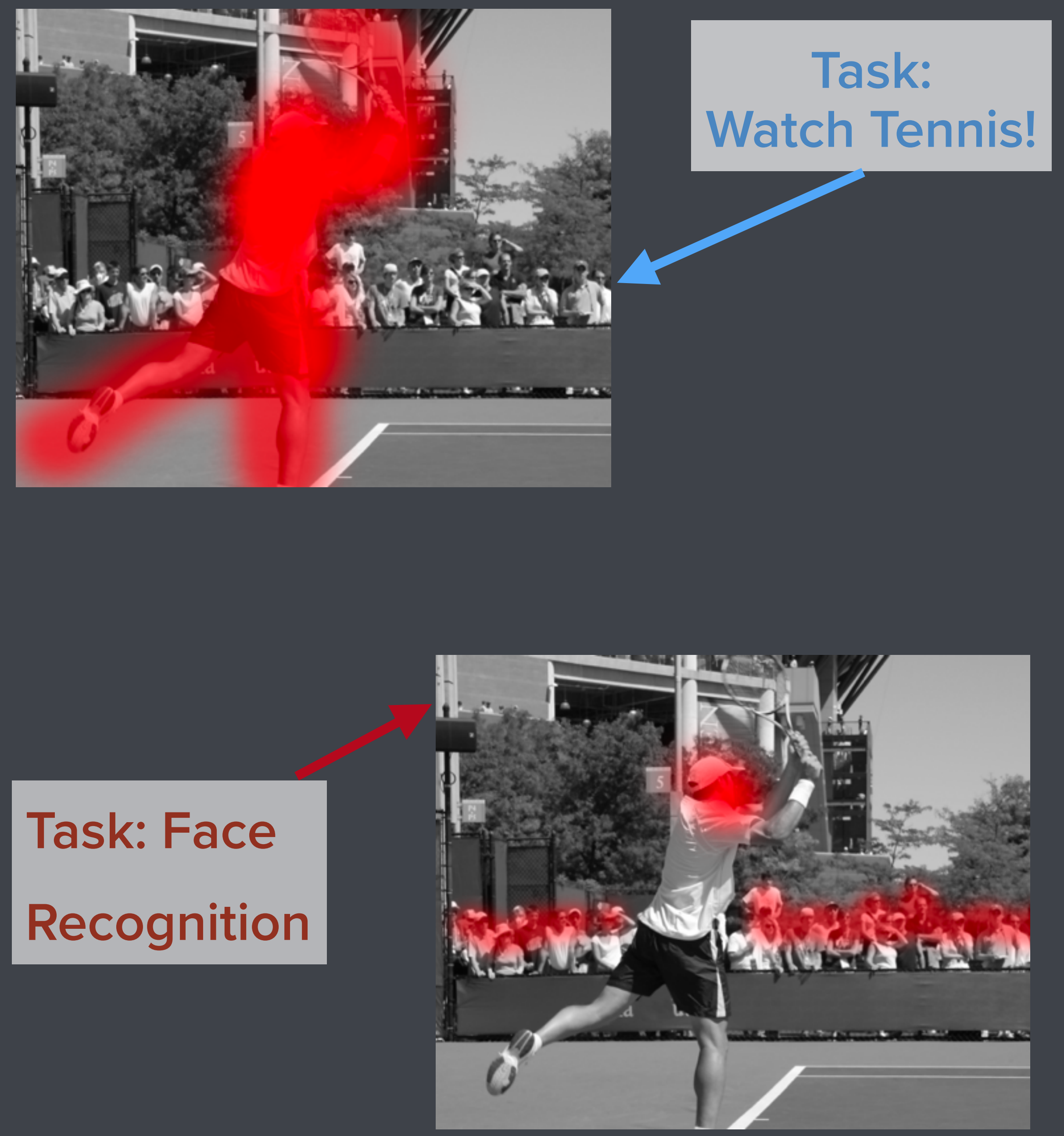
Custom codec for each domain

Content-aware



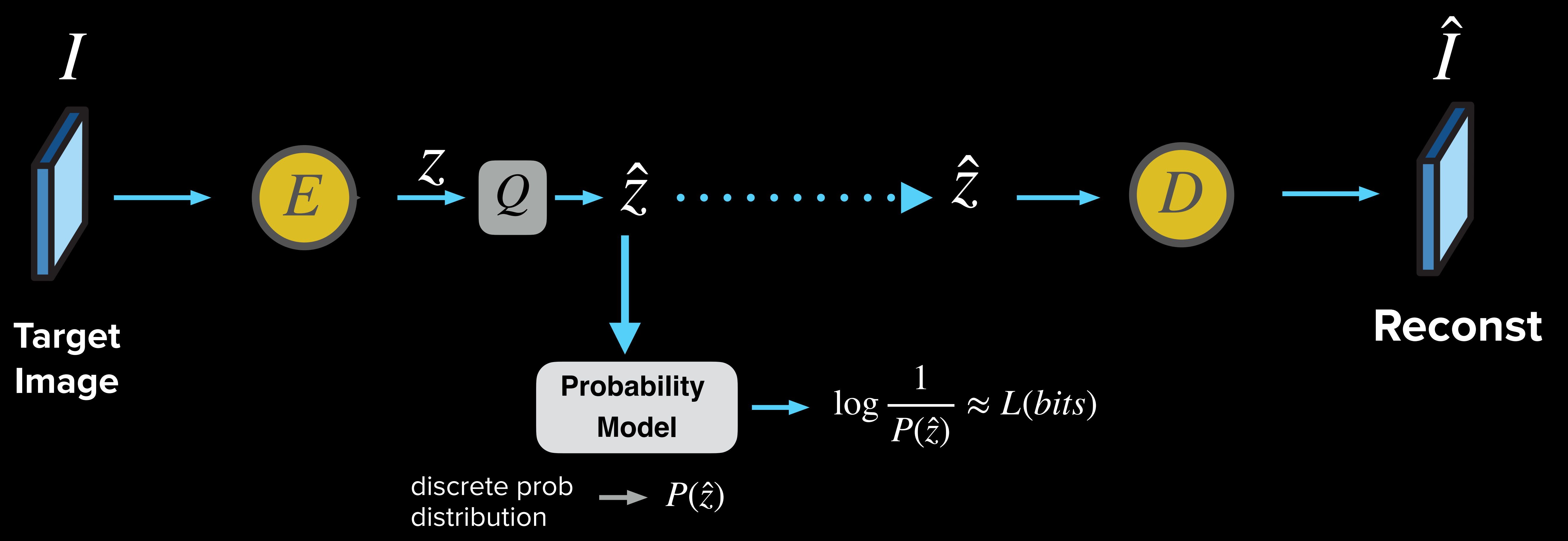
Focus on areas of high importance

Task-aware



Computation on compressed representation

Learned Image Codecs



Loss Function = $\log \frac{1}{P(\hat{z})} + \lambda d(I, \hat{I})$

Possible to train end-to-end!

No hand-tuning parameters

Can "learn" the parameters

Better distortion-Model separation

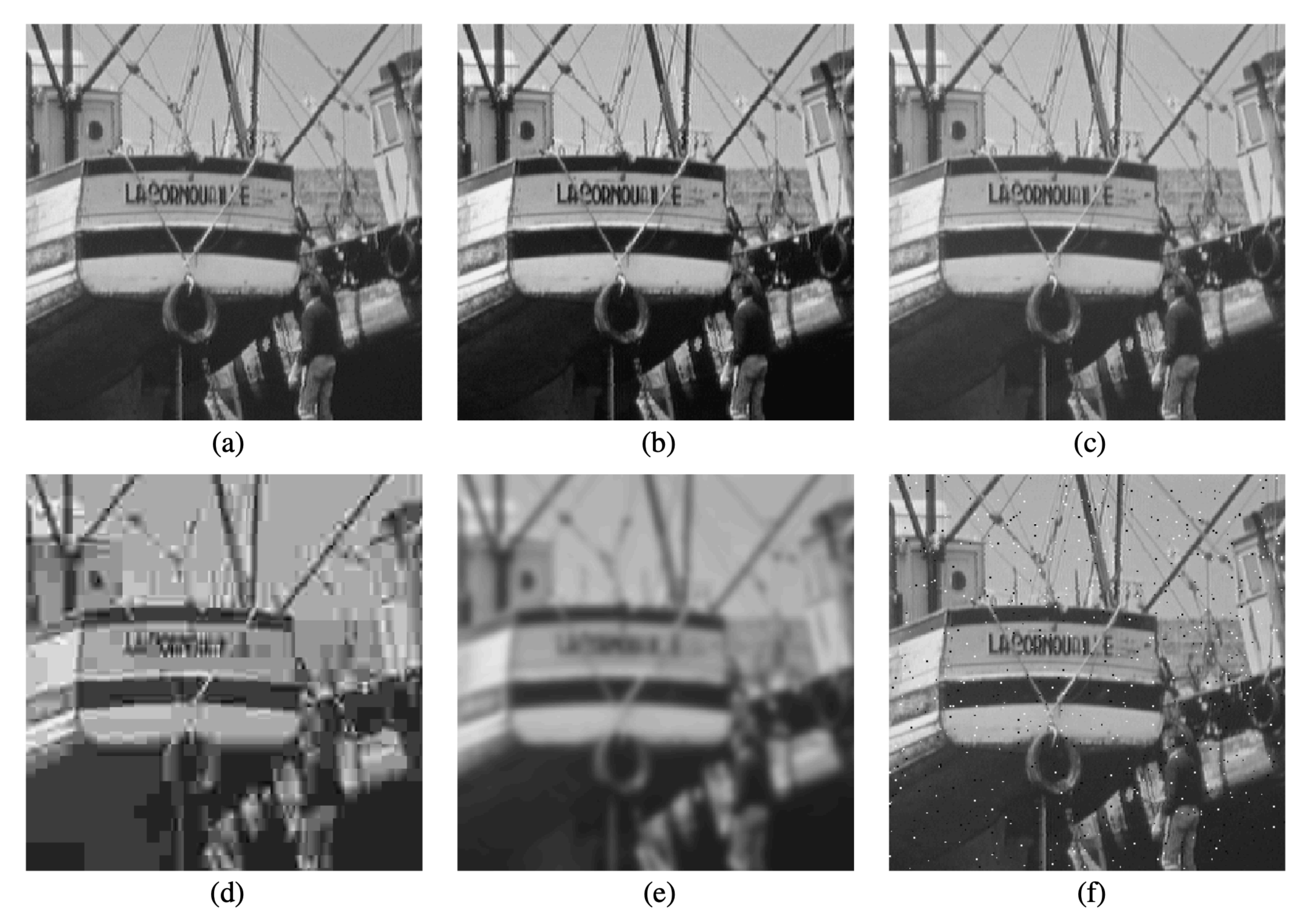
Easy to tune model to different distortion metrics

Given source image (a) which of the following images do you prefer visually?

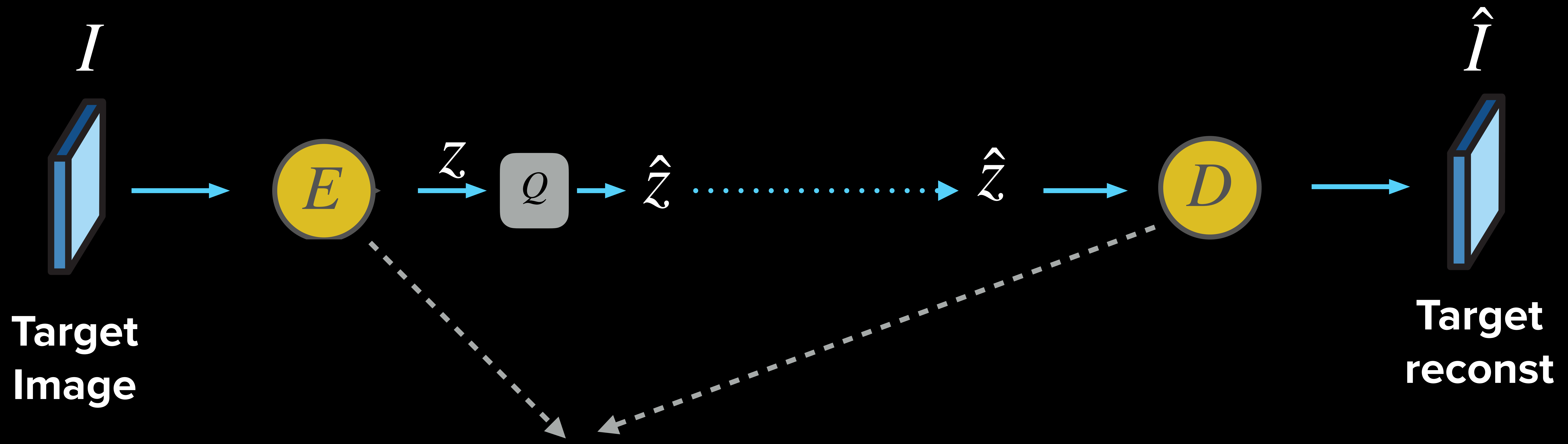
(b), (c), (d), (e), (f)

Given source image (a) which of the following images does a compressor with MSE distortion prefer?

(b), (c), (d), (e), (f)

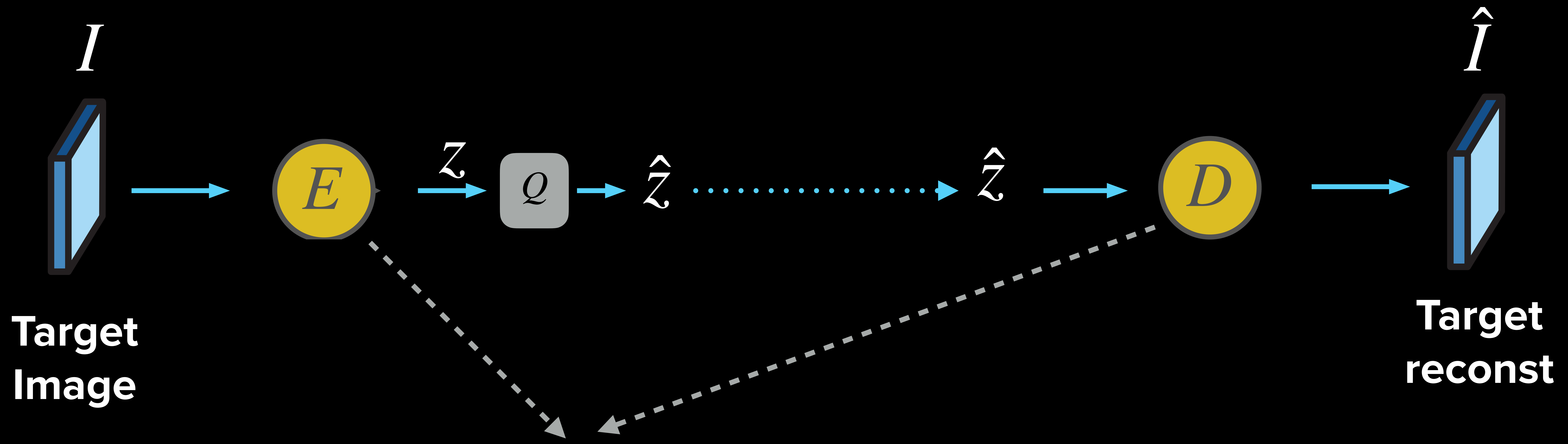


Design Decisions: (i) Backbones



- **Question:** How can the same model work for any image sizes?

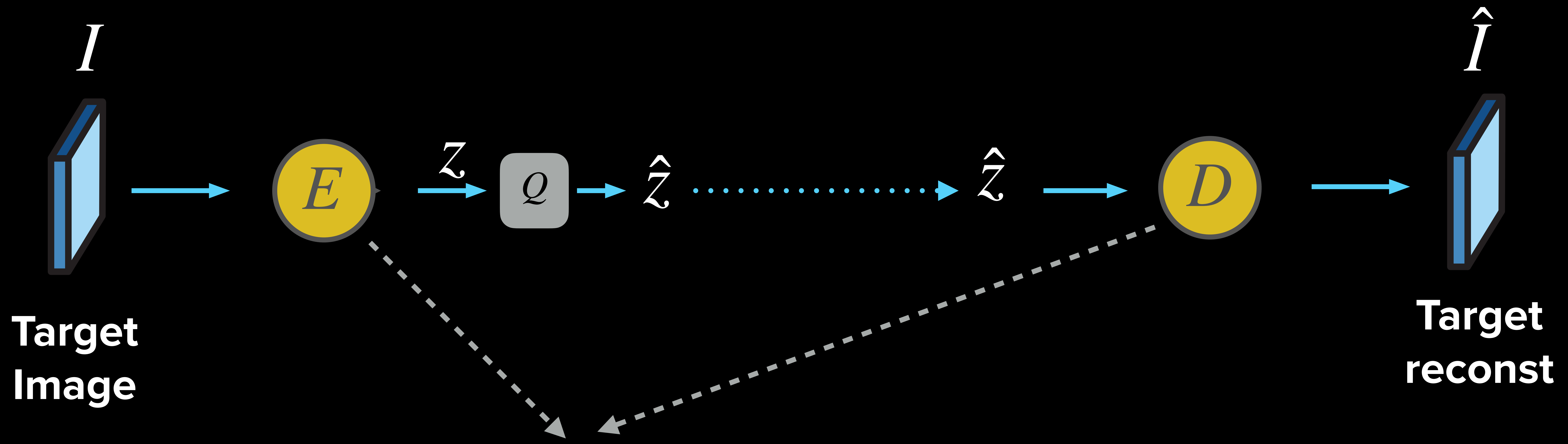
Design Decisions: (i) Backbones



- **Question:** How can the same model work for any image sizes?

  - Only use Conv, Deconv … (no Fully Connected Layers)

Design Decisions: (i) Backbones

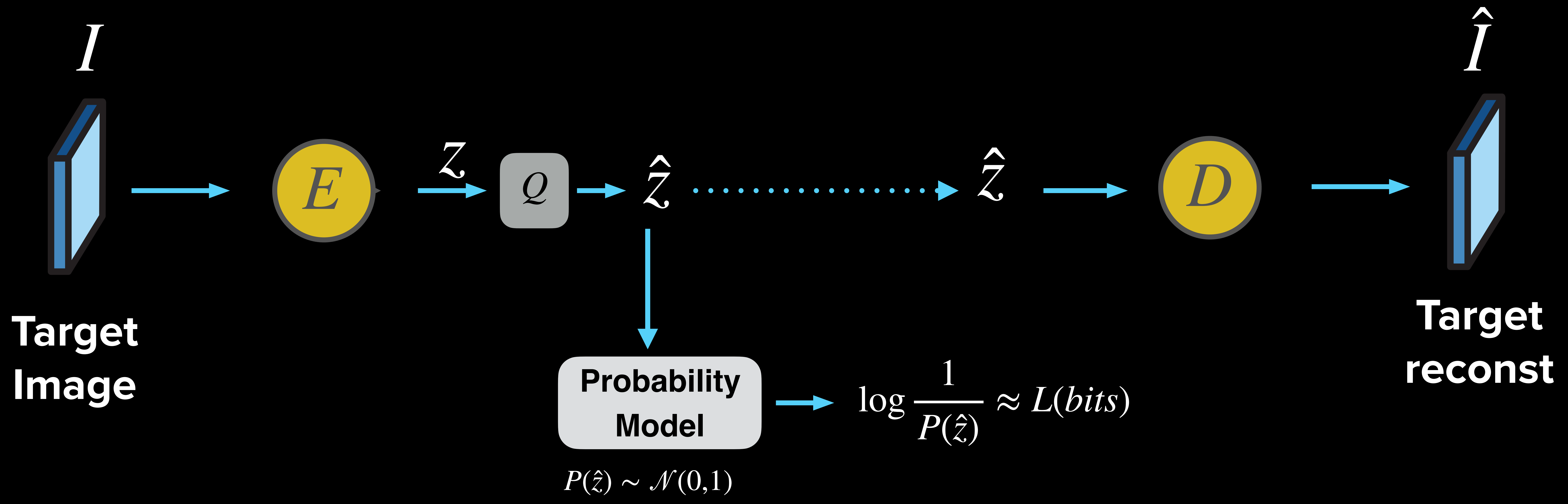


- **Other Improvements:**

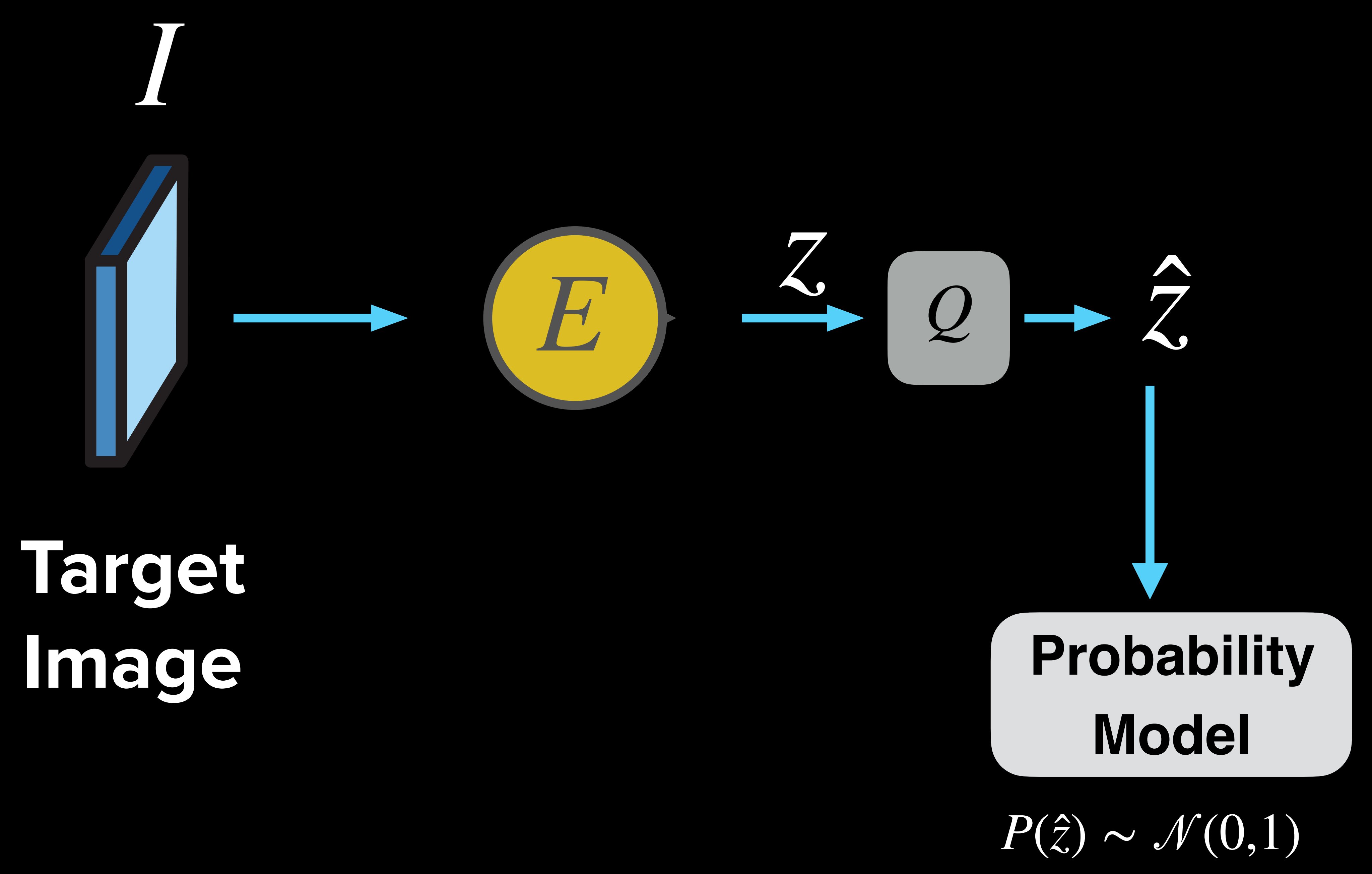
  - Multiscale Encoder/Decoder: [Rippel et. al. 2018]

  - using GDN non-linearity: [Balle, 2017/18]

Design Decisions: (ii) Probability Model



Design Decisions: (ii) Probability Model

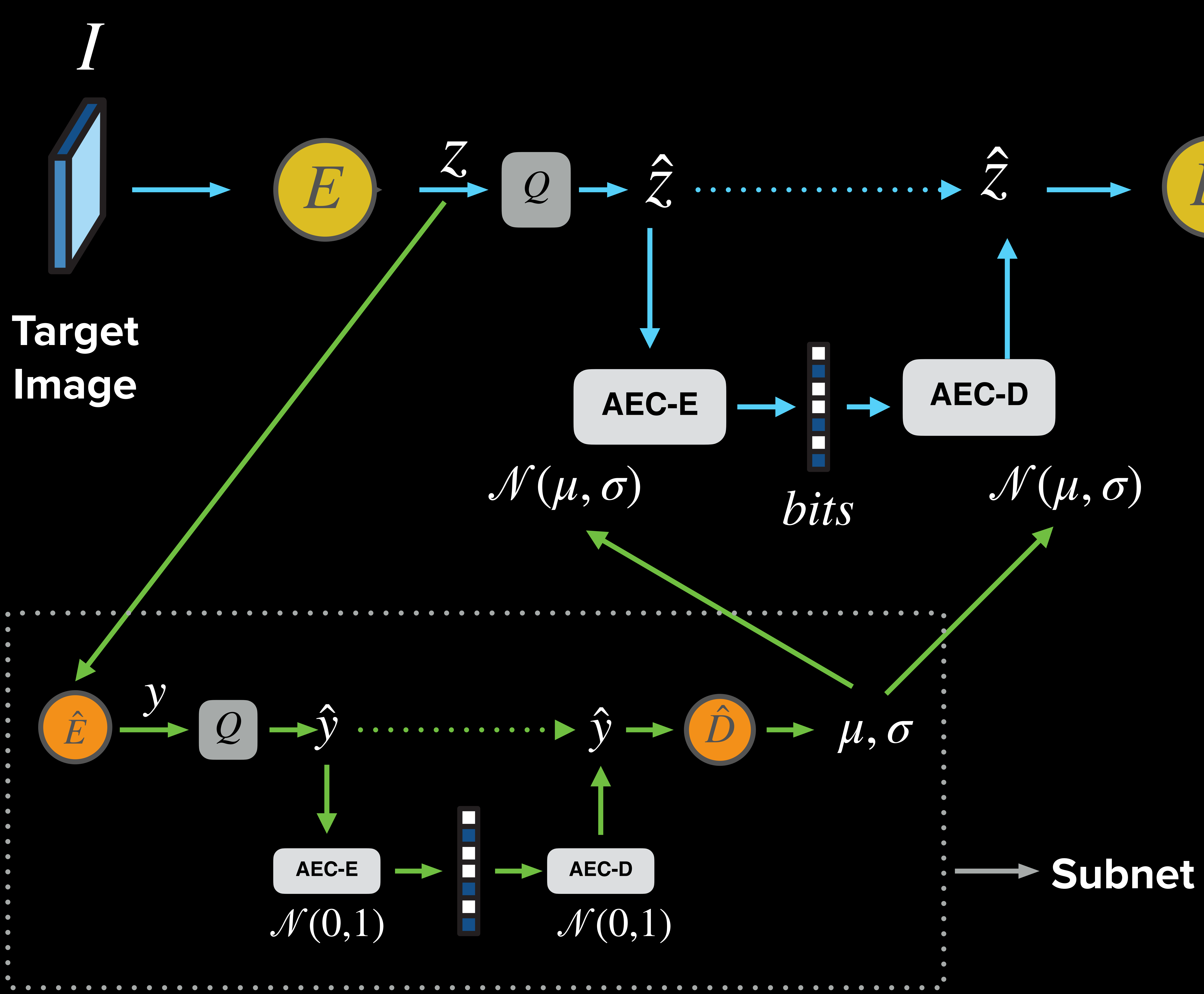


- More complex Probability models
- $PDF(\hat{z}_i) \rightarrow \mathcal{N}(\mu_i, \sigma_i)$,

  i.e: μ, σ are different per element of $\hat{z}$
- Need to now encode μ, σ tensors:

  - *Hyperprior* approach

    [Balle, ICLR18]



I
Target Image
E
z
Q
$\hat{z}$
$\hat{z}$
D
$\hat{I}$
Target reconst
AEC-E
AEC-D
$\mathcal{N}(\mu, \sigma)$
bits
$\mathcal{N}(\mu, \sigma)$
$\hat{E}$
y
Q
$\hat{y}$
$\hat{y}$
$\hat{D}$
μ, σ
AEC-E
AEC-D
$\mathcal{N}(0,1)$
$\mathcal{N}(0,1)$
Subnet

Design Decisions: (ii) Probability Model

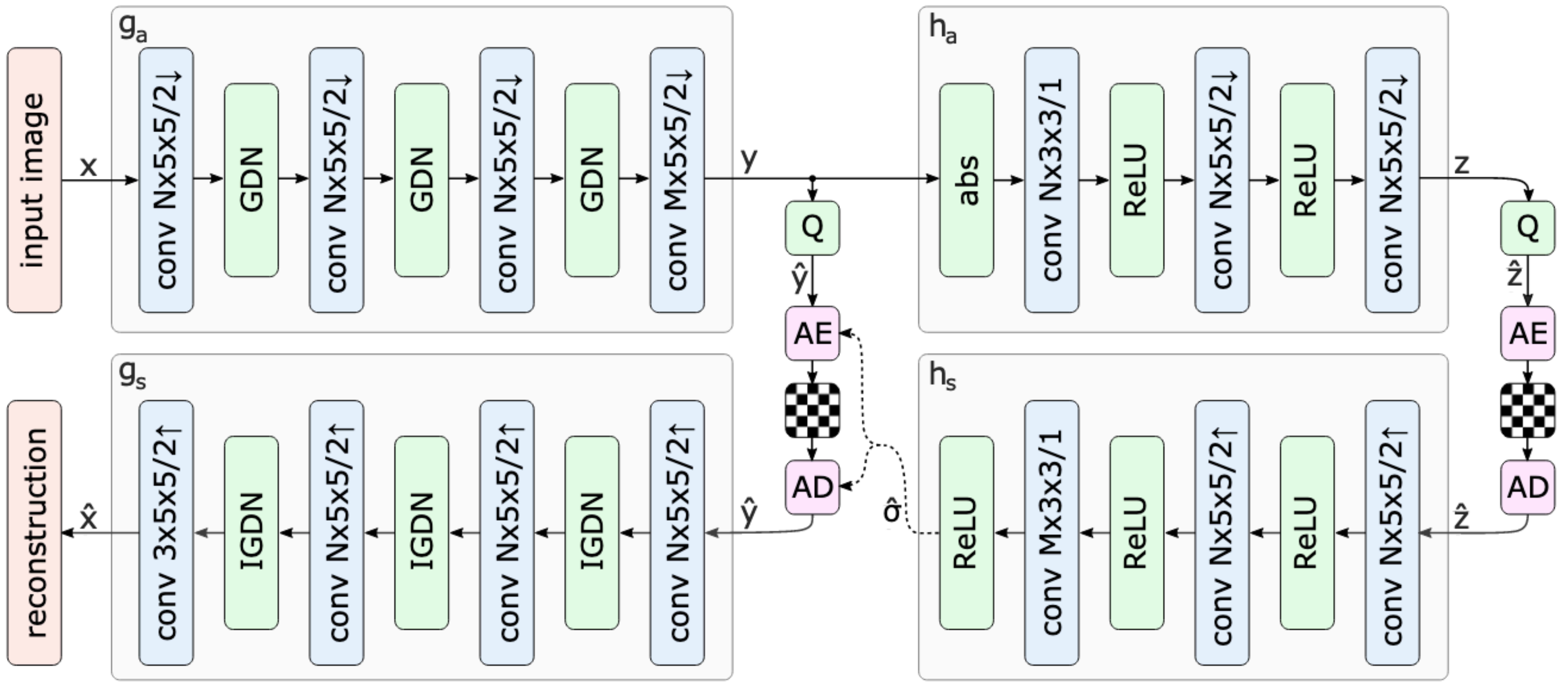


Figure 4: Network architecture of the hyperprior model. The left side shows an image autoencoder architecture, the right side corresponds to the autoencoder implementing the hyperprior. The factorized-prior model uses the identical architecture for the analysis and synthesis transforms g_a and g_s. Q represents quantization, and AE, AD represent arithmetic encoder and arithmetic decoder, respectively. Convolution parameters are denoted as: number of filters $\times$ kernel support height $\times$ kernel support width / down- or upsampling stride, where $\uparrow$ indicates upsampling and $\downarrow$ downsampling. N and M were chosen dependent on λ, with $N = 128$ and $M = 192$ for the 5 lower values, and $N = 192$ and $M = 320$ for the 3 higher values.

Variational image compression with a scale hyperprior

Balle et.al. 2018

Design Decisions: (ii) Probability Model

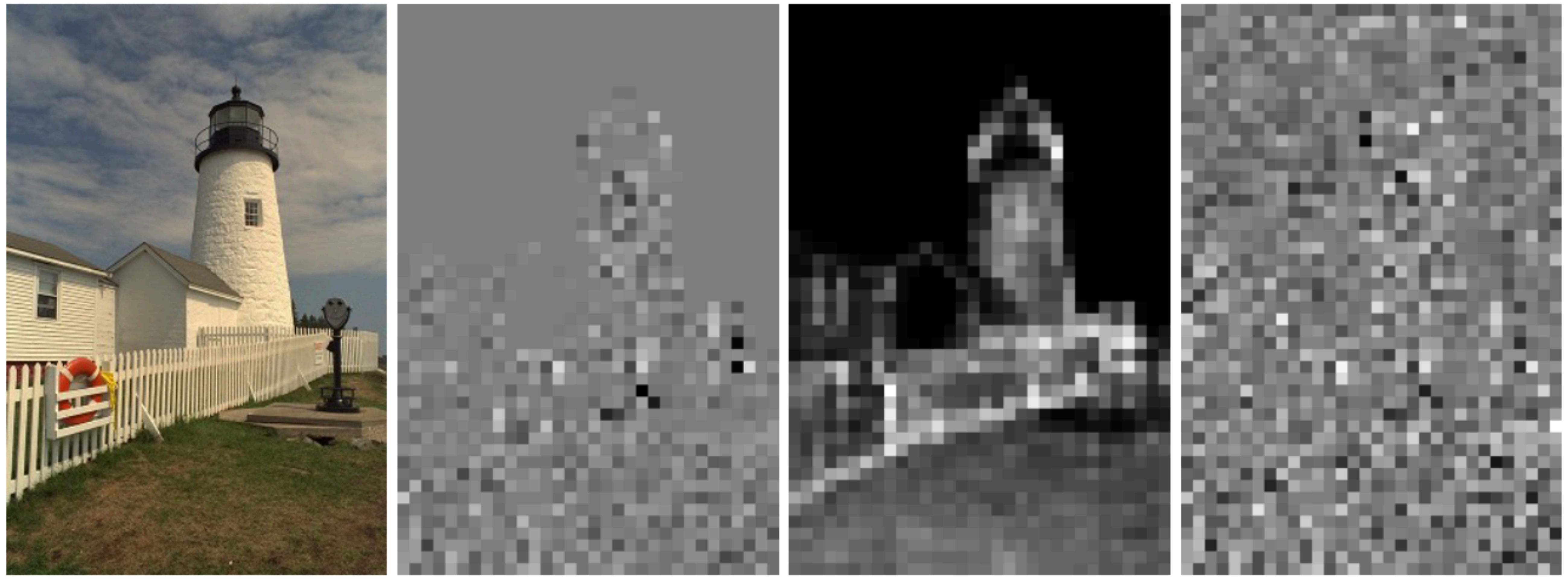
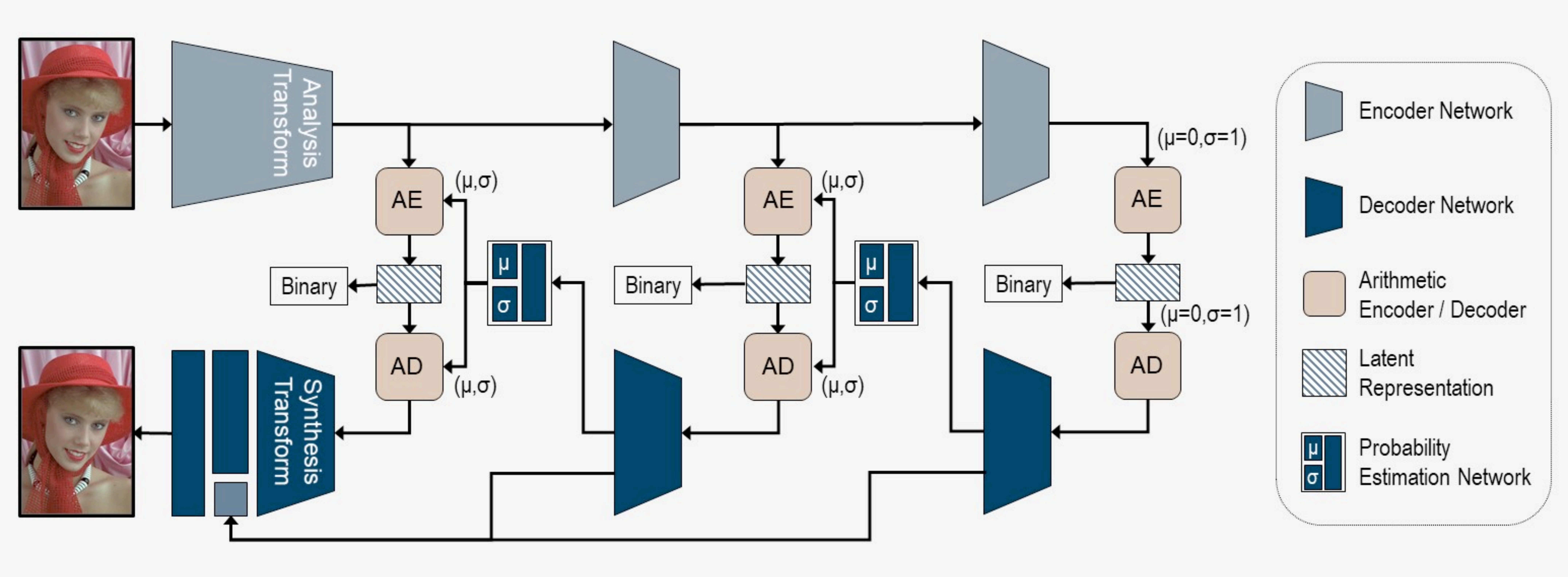


Figure 2: Left: an image from the Kodak dataset. Middle left: visualization of a subset of the latent representation $\boldsymbol{y}$ of that image, learned by our factorized-prior model. Note that there is clearly visible structure around edges and textured regions, indicating that a dependency structure exists in the marginal which is not represented in the factorized prior. Middle right: standard deviations $\hat{\boldsymbol{\sigma}}$ of the latents as predicted by the model augmented with a hyperprior. Right: latents $\boldsymbol{y}$ divided elementwise by their standard deviation. Note how this reduces the apparent structure, indicating that the structure is captured by the new prior.

Design Decisions: (ii) Probability Model



Hyper-hyper-prior models:
https://huzi96.github.io/coarse-to-fine-compression.html

Benchmarks

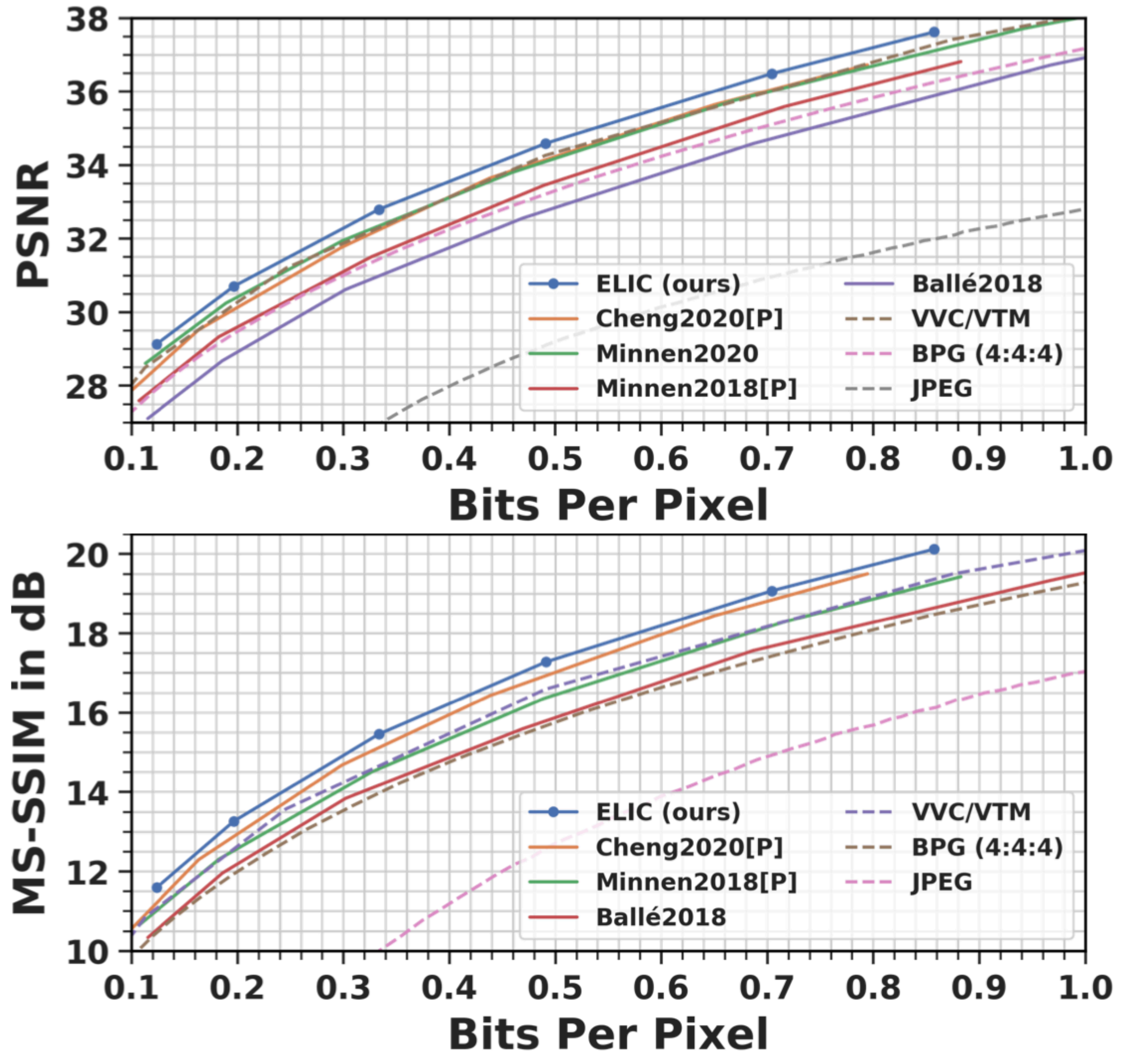


Figure 9. Rate-distortion curves of various image compression approaches. The results are evaluated on Kodak. All shown learned models are optimized for minimizing MSE.

ELIC: Efficient Learned Image Compression with Unevenly Grouped Space-Channel Contextual Adaptive Coding
He et.al. 2022

Benchmarks

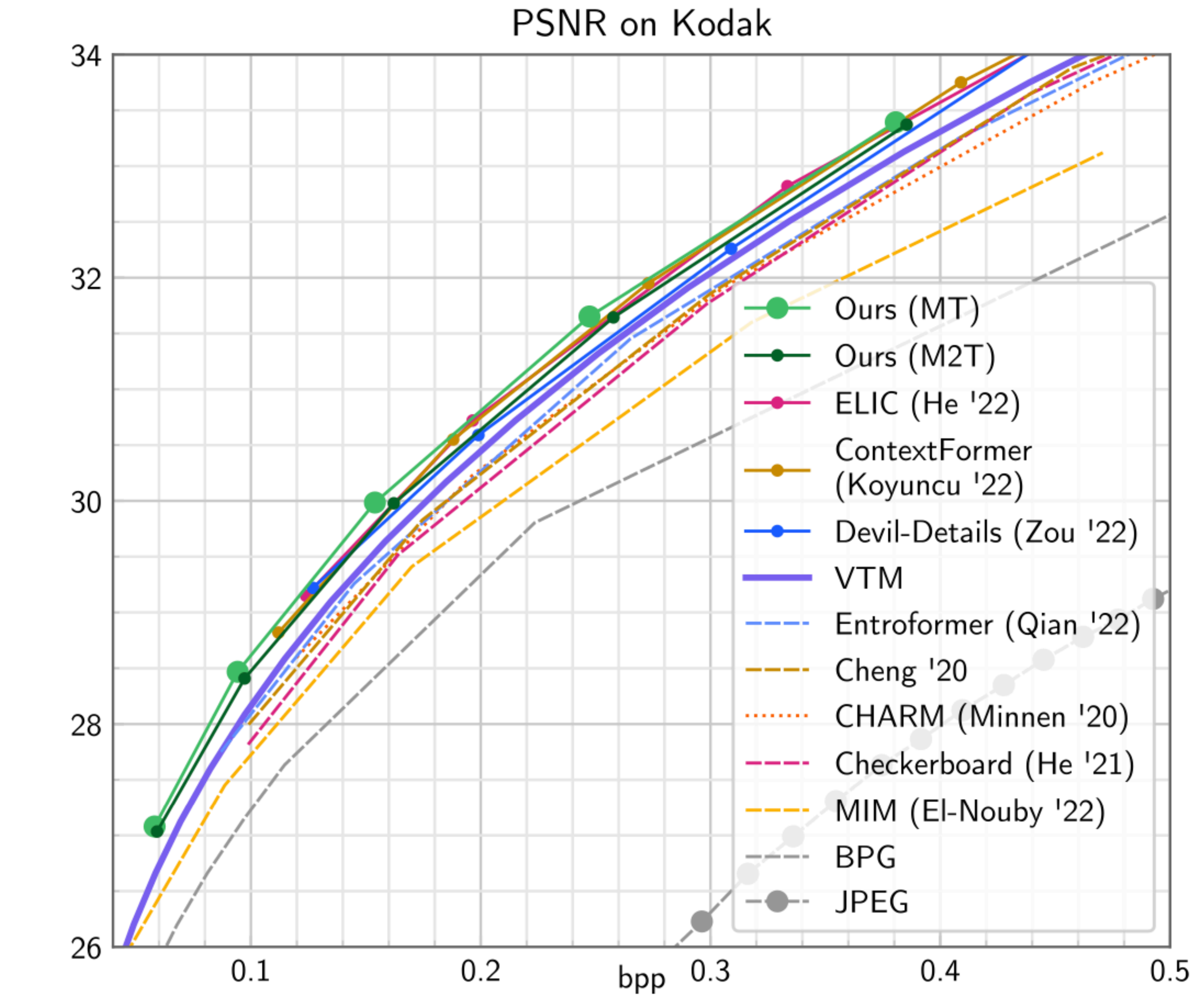
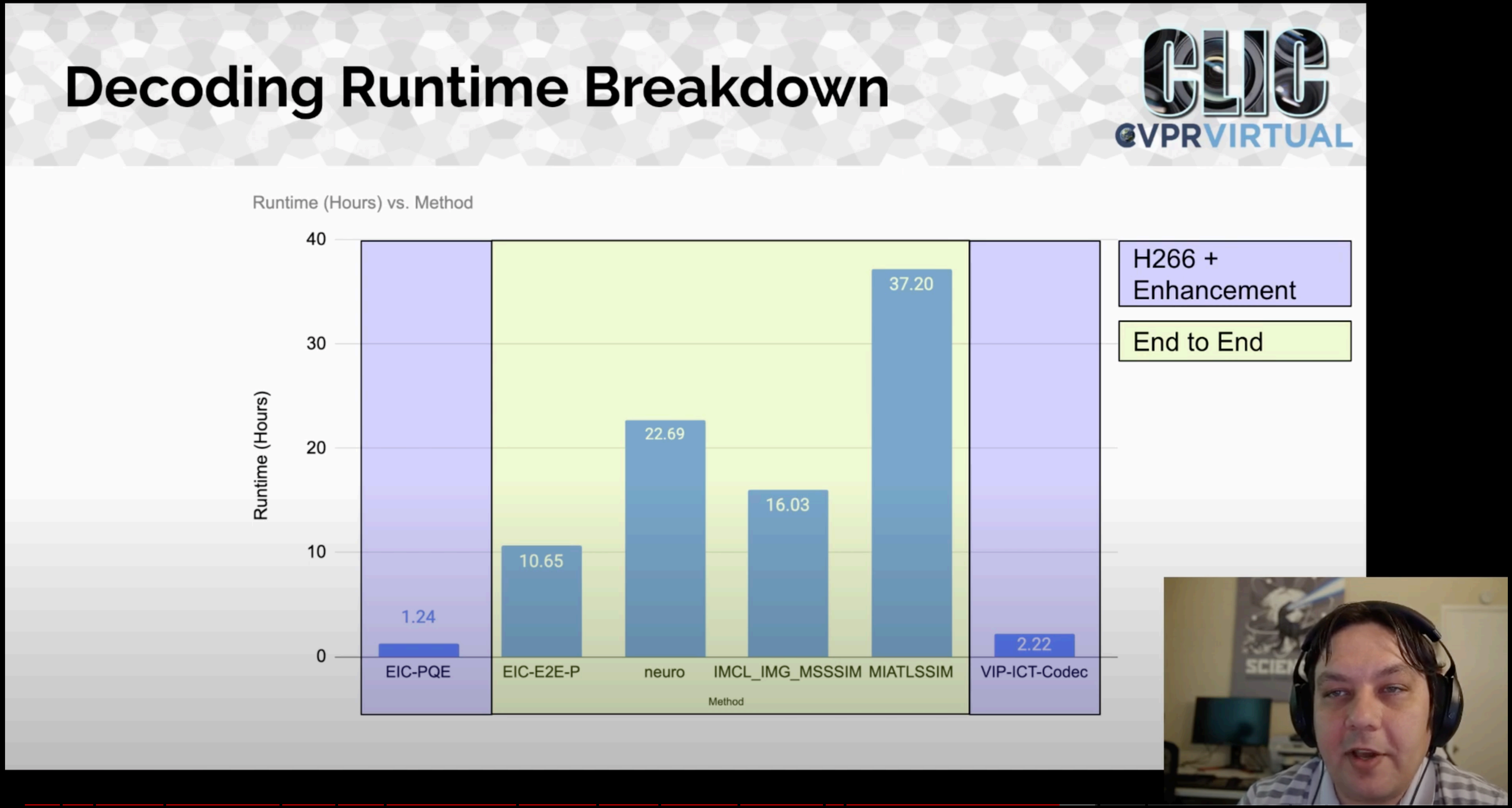


Figure 1: Rate distortion results on Kodak. Our *MT* outperforms the prior state-of-the-art ELIC [18]; *M2T* only incurs a small reduction in rate-distortion performance compared to MT while running about $4\times$ faster on hardware (see Fig. 4)

M2T: Masking Transformers Twice for Faster Decoding

Mentzer et.al. 2023

Issues: (i) Speed!



Slide from Dr. Toderici's talk at CVPR20

Issues: (i) Speed!

- **Design decisions:** Need to choose frameworks, which aren’t fundamentally slow/causal (eg: avoid autoregressive image compression methods)

Issues: (i) Speed!

- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

- **For example:** [ELF-VC, Rippel et.al. 2021, ArXiv]

  real-time HD 720 decode on mid-range GPU

  - VGA 640x480: encode @ 47 FPS, decode @ 91 FPS

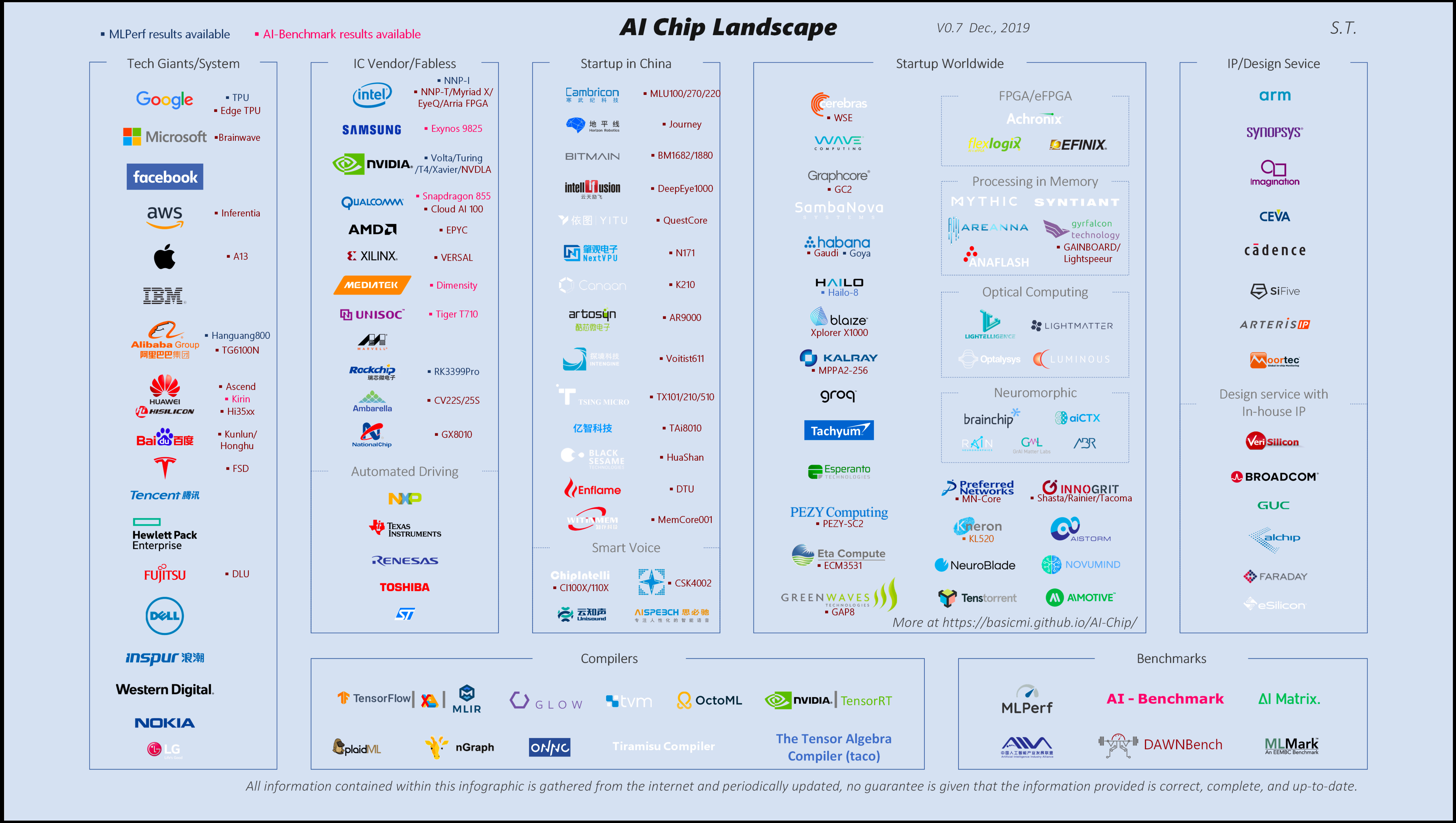
  - HD 1280x720: encode @ 19 FPS, decode @ 35 FPS

Issues: (i) Speed!

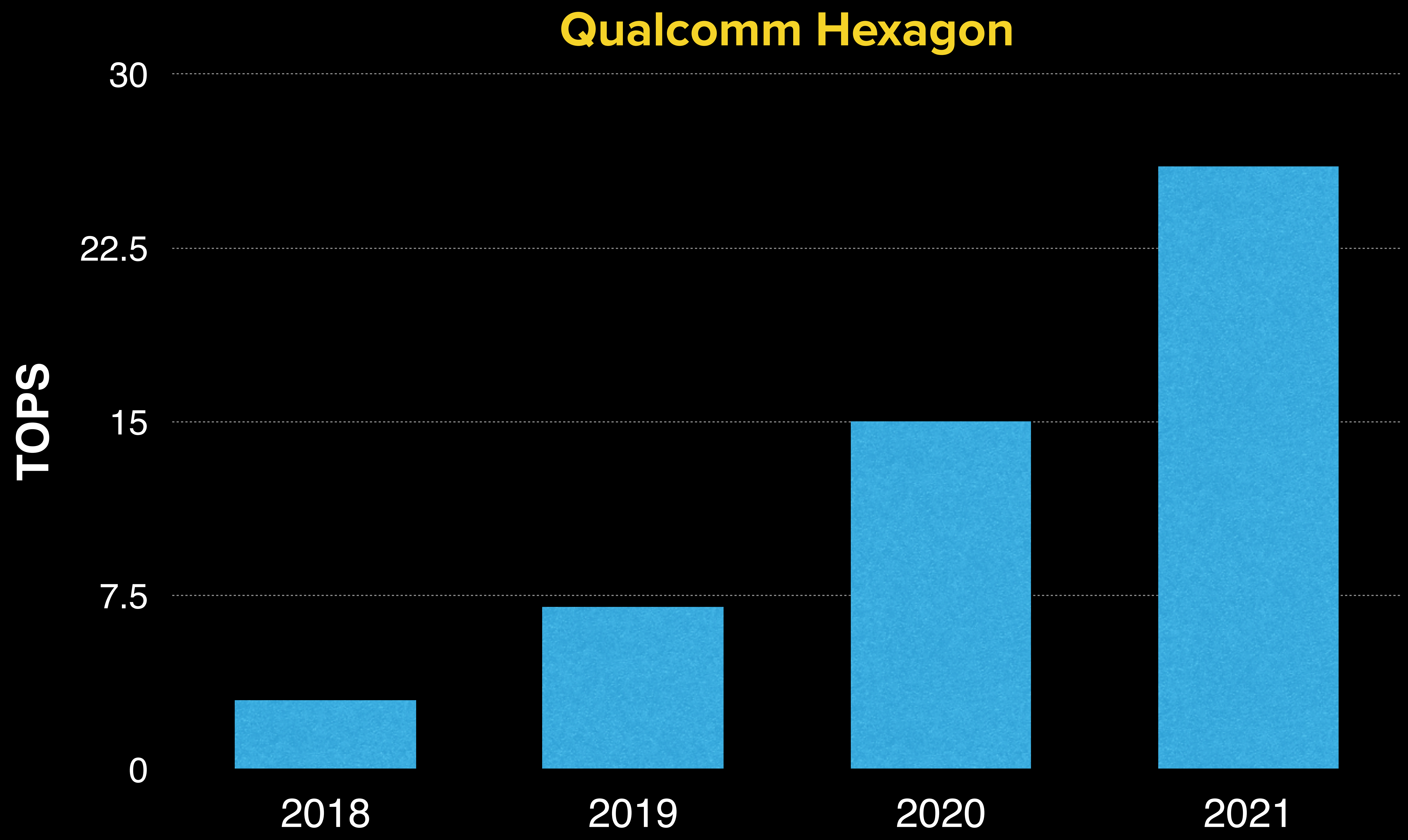
- **Design decisions:** Need to choose frameworks, which aren't fundamentally slow/causal (eg: avoid autoregressive image compression methods)

- **Faster Hardware:** Hardware support for NN keeps improving year by year

Issues: (i) Speed!



Issues: (i) Speed!

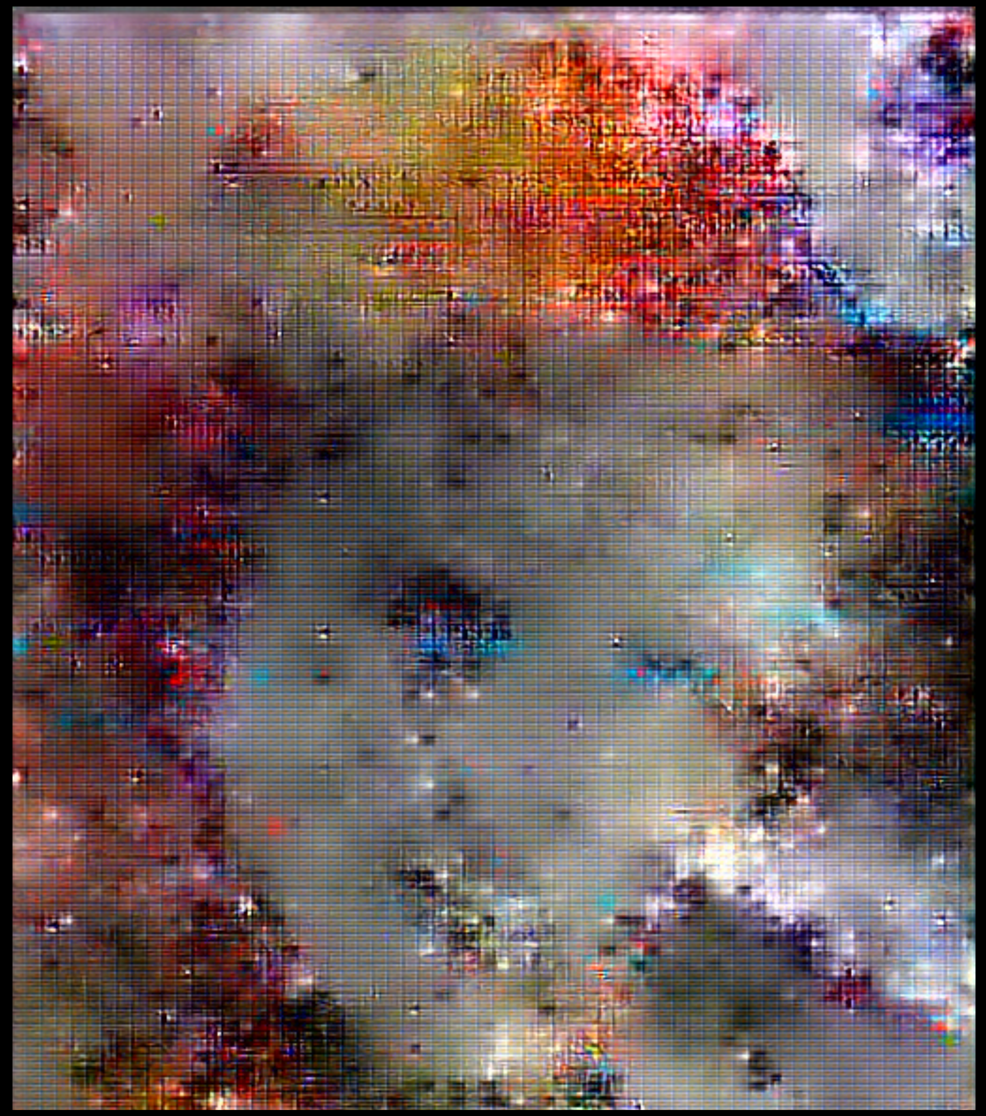


Issues: (ii) Determinism!

Different hardware =>

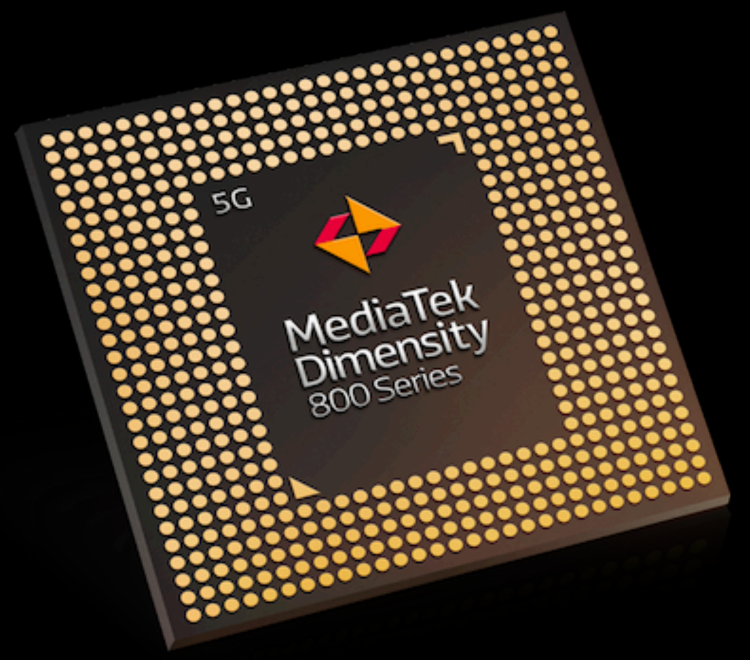
different floating point implementation =>

catastrophic failures!



Issues: (ii) Determinism!

- Encoding/decoding must yield exactly the same outputs, irrespective of hardware architecture



- Floating point (FP32/16) models don't work: Model Quantization necessary [E.g: Ballé 2019]

Take-aways from Today

ML-based codecs allow for ***learned encoder-decoder transforms*** and therefore

(i) better fidelity to chosen probability model over latents => better rate-distortion

(ii) allow for substituting distortion of the choice

(iii) domain adaptable and flexible

Main idea to achieve ML-based codec is to ***overcome (automatic) differentiation*** over

(i) quantization and (ii) discrete probability models

ML-based methods will likely form the ***basis of future compression***