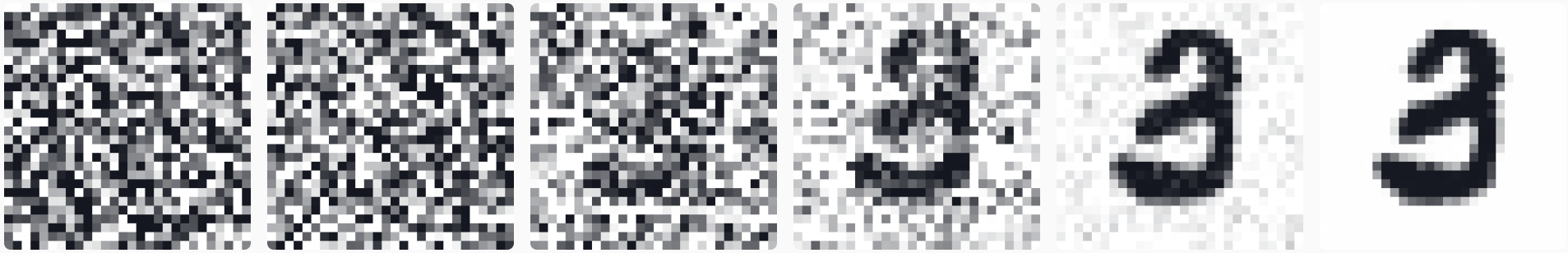
OUR MODEL

From noise to images

Denoising diffusion, from one noisy pixel to text-to-image models



t = 1000 600 400 200 50 0

One sample from the small model we train in this lecture: 814,145 parameters, about 50 minutes on a laptop GPU.

START WITH A QUESTION

This digit was pure noise 1,000 steps ago



What happened at each step: one network looked at the current image and removed a *little* of what it thought was noise.

What never happened: nobody gave it a digit to copy. It started from random numbers.

How can learning to remove noise let us create an image from nothing but noise?

THE SAME IDEA, CONDITIONED ON DIFFERENT THINGS

PUBLISHED

What would each model produce?

a sentence

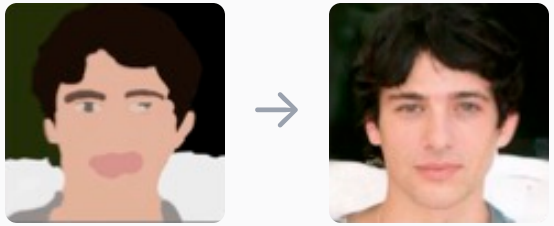
"A basket full of cherries"



a face with a region missing



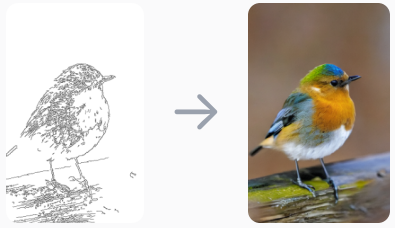
a crude painting



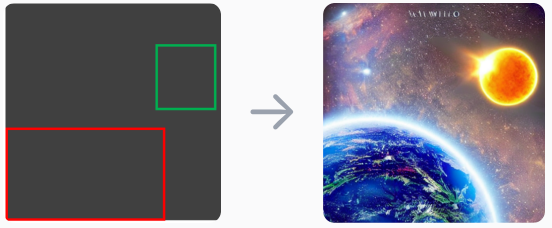
a blurry, low-resolution face



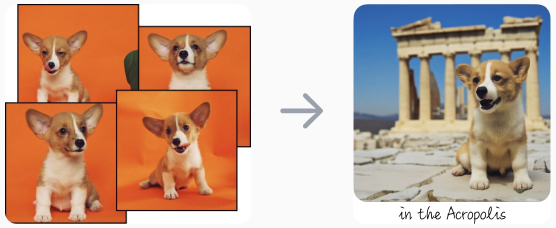
an edge map + "bird"



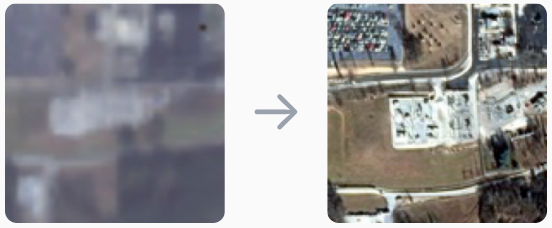
boxes: planet, sun



4 photos of one dog + "at the Acropolis"



a 10 m satellite image



> **Eight different inputs, one recipe. By the end you will be able to say, for each, what the network was trained to predict.**

LDM (Rombach et al. 2022) · RePaint (Lugmayr et al. 2022) · SDEdit (Meng et al. 2022) · SR3 (Saharia et al. 2022) · ControlNet (Zhang et al. 2023) · GLIGEN (Li et al. 2023) · DreamBooth (Ruiz et al. 2023) · DiffusionSat (Khanna et al. 2024)

PUBLISHED

One prompt, several valid images

“A basket full of cherries”



sample 1



sample 2



sample 3



sample 4

Look closely: two samples carry stock-photo watermarks. The model was trained on web images with captions, and it learned the watermarks too.

> **Which one is correct? None and all: a prompt describes a *distribution* of images, and a generator samples from it.**

Latent Diffusion, Rombach et al. 2022, project-page Figure 17 (model trained on Conceptual Captions). Seeds are not reported.

COMPLETION · WE WILL DO THIS OURSELVES IN PART 6

PUBLISHED

Fill a missing region



input · lavender = missing



> **The visible pixels constrain the answer; they do not decide it. Smile or no smile is still the model's choice.**

RePaint, Lugmayr et al. 2022 (project figure): one input, five independent samples.

PUBLISHED

Is the sharp detail real?



Bicubic = the low-resolution input, enlarged smoothly. Regression = one network trained with squared error.

> **Regression blurs, a generator invents. Compare the wrinkles under the eye: plausible, and different from the real photo.**

SR3, Saharia et al. 2022 (project figure). The lower row enlarges the boxed eye region.

A QUESTION FOR OUR LAB

PUBLISHED

Would you count brick kilns in a generated image?

Sentinel-2, 10 m per pixel	DiffusionSat sample	real high-resolution image
		
the input	conditioned on the input	withheld reference

Our lab maps brick kilns across South Asia from this kind of 10 m imagery. A kiln is only a few pixels wide.

> **Plausible is not measured. Before using generated detail, we must check it against held-out observations.**

DiffusionSat, Khanna et al. 2024, super-resolution figure (left panel, row 5). Columns: Sentinel-2 RGB, DiffusionSat output, high-resolution reference.

Two generators we already trained, and their problems

VAE samples (16-number code)



One decoder pass from a random code. Smooth to train, but the digits come out **blurry**.

GAN samples



Sharper, but training is a **two-player game** that can collapse onto a few modes.

> **Diffusion's bet: train with a plain regression loss, like the VAE, and generate in many small steps instead of one big jump.**

THE PLAN

Four questions answer the core of the lecture

1. What noisy input do we train on? — Part 1 · corrupt a digit
2. What does the network learn to predict? — Part 2 · training
3. How do predictions become a new image? — Part 3 · sampling
4. Why is that the right loss? — Part 4 · Lecture I's ELBO

Then: inside the network (Part 5), steering it with a prompt (Part 6), and text instead of pixels (Part 7).

One running example throughout: a single 7 from the MNIST test set, and one pixel of it.

PART 1 OF 7

Destroy a digit, one small step at a time

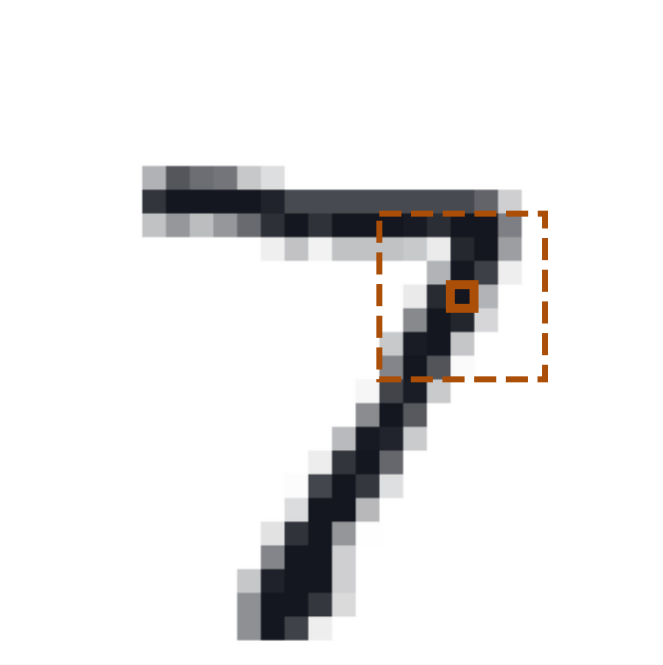
Before any learning, we fix a recipe that turns an image into noise. **Goal of this part:** a formula for the noisy image at any step t, checked by hand on one pixel. No neural network yet.

1 **Corrupt a digit**
2 Train: predict the noise
3 Sample: run it backwards
4 Why this loss: the ELBO
5 Inside the network
6 Steer with a condition
7 Text by denoising

THE RUNNING EXAMPLE

MNIST TEST IMAGE 0

Our 7, and one pixel of it



254	250	229	254	254	140	0
67	59	21	236	254	106	0
0	0	83	253	209	18	0
0	22	233	255	83	0	0
0	129	254	238	44	0	0
59	249	254	62	0	0	0
133	254	187	5	0	0	0

Pixels are stored as 0 … 255. We rescale them:

$$x = \frac{v}{127.5} - 1$$

0 (paper) → **−1**, **255** (ink) → **+1**

our pixel row 12, column 19: value 255, so $x_0 = +1$.

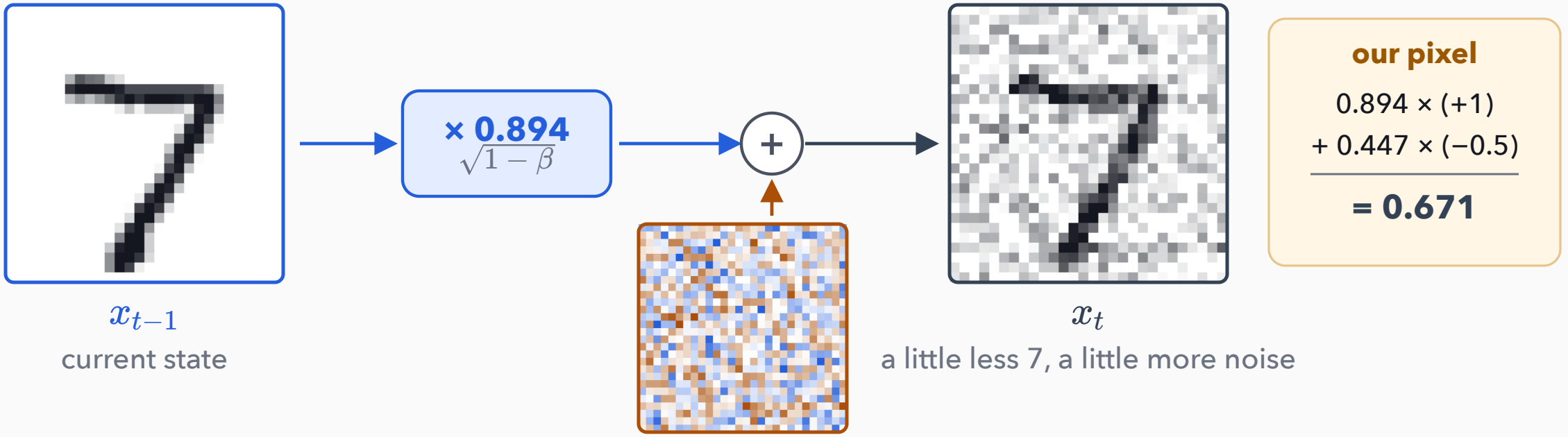
One image is a tensor $[1, 28, 28]$; a batch is $[B, 1, 28, 28]$.

We will follow this one pixel, with value +1, through every equation in Parts 1–3.

THE FORWARD STEP

COMPUTED β = 0.2 so you can see it

One step: shrink the image a little, then add fresh noise



$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \eta_t, \qquad \eta_t \sim \mathcal{N}(0, I)$$

With β = 0.2, our pixel goes from 1 to 0.671. Each step keeps most of the image and mixes in a little noise.

WHY THE SQUARE ROOTS?

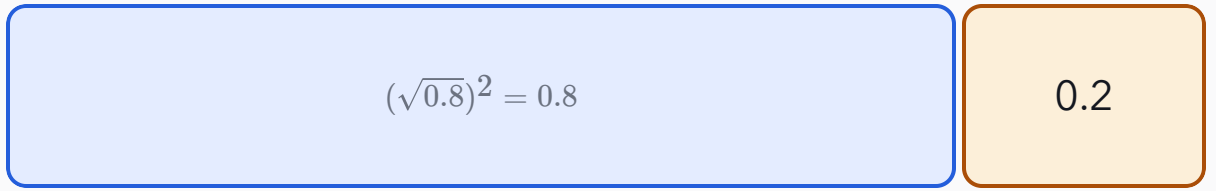
COMPUTED

Variances add, so scale by square roots

a unit-variance signal



after one step, β = 0.2



total **1.0**

without the roots: 0.8·x + 0.2·η



total **0.68**, and shrinking every step

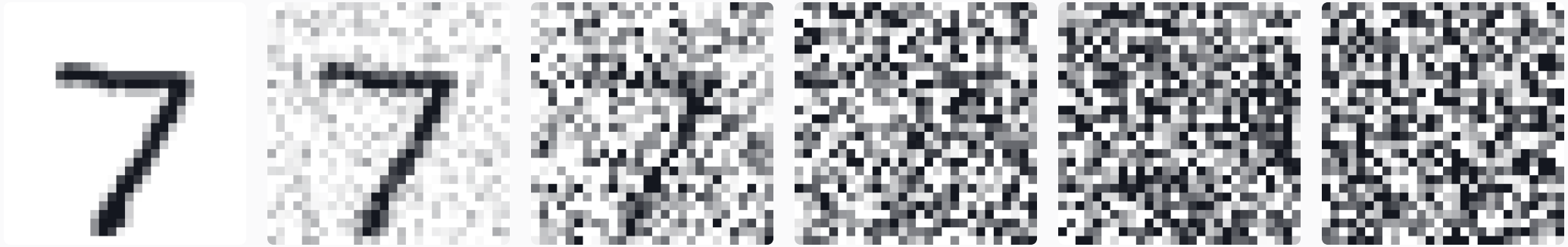
$$\mathrm{Var}(x_t) = (1-\beta_t)\,\mathrm{Var}(x_{t-1}) + \beta_t = 1 \quad \text{if} \quad \mathrm{Var}(x_{t-1}) = 1$$

> **The square roots keep the total variance at 1, so after many steps we land exactly on a standard Gaussian, $\mathcal{N}(0, I)$.**

REPEAT ON THE REAL SCHEDULE

COMPUTED no model

A thousand small steps dissolve the 7



t = 0
signal 1.00

t = 100
signal 0.947

t = 300
signal 0.63

t = 500
signal 0.28

t = 700
signal 0.083

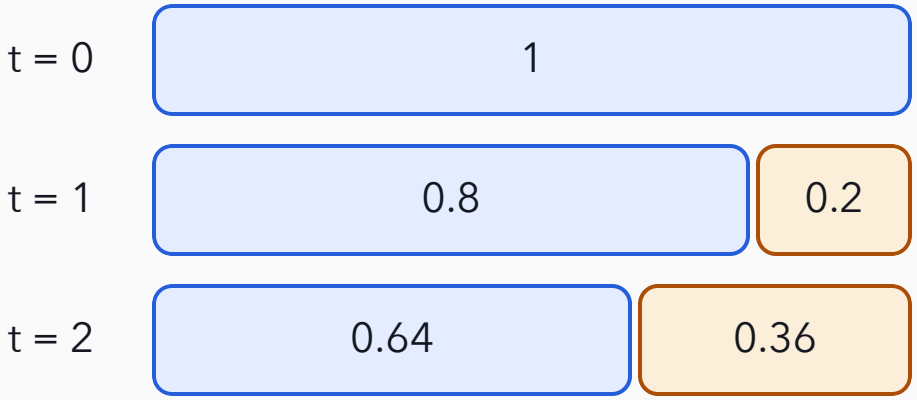
t = 1000
signal 0.006

One fresh noise image per step; β grows linearly from 0.0001 to 0.02 over T = 1000 steps. “Signal” is the number that multiplies the original 7 after t steps.

After about t = 600 every panel looks the same: almost nothing of the 7 is left.

COMPUTED

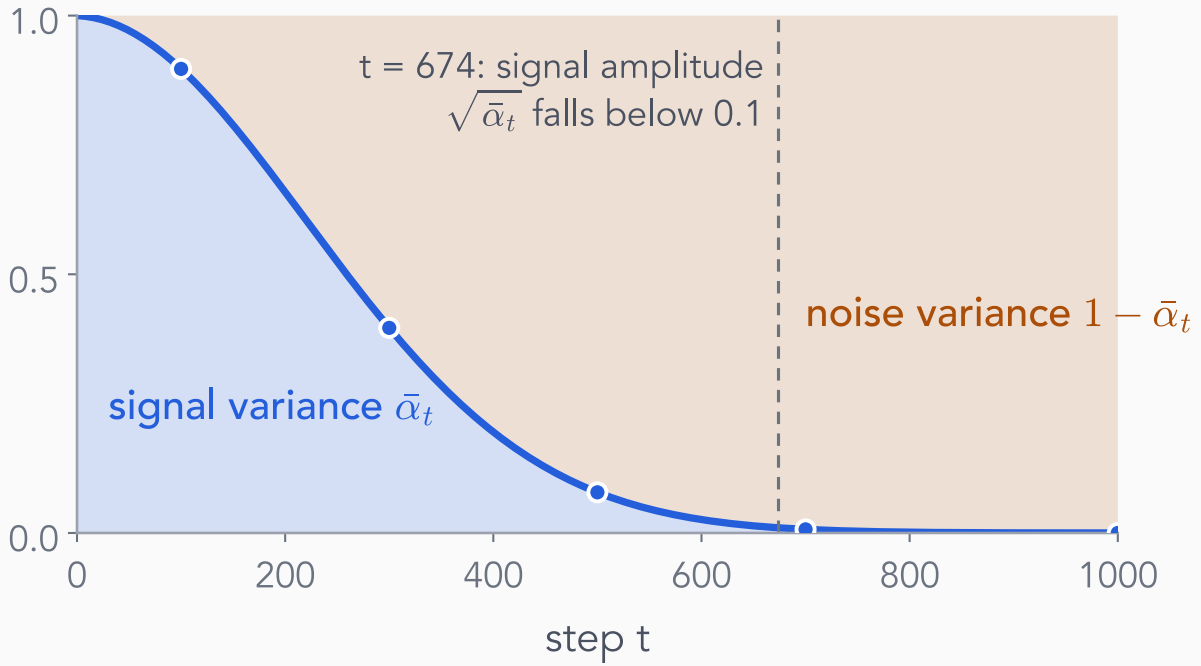
How much signal is left after t steps: $\bar{\alpha}_t$



$$\alpha_t = 1 - \beta_t \quad (\text{here } 0.8)$$

$$\bar{\alpha}_t = \alpha_1 \alpha_2 \cdots \alpha_t$$

$$\bar{\alpha}_2 = 0.8 \times 0.8 = 0.64$$



The real schedule: the dots are the panels on the previous slide.

Blue is signal variance $\bar{\alpha}_t$, amber is noise variance $1 - \bar{\alpha}_t$; they always add to 1. Signal falls below half at t = 260.

A SHORTCUT WE WILL USE IN TRAINING

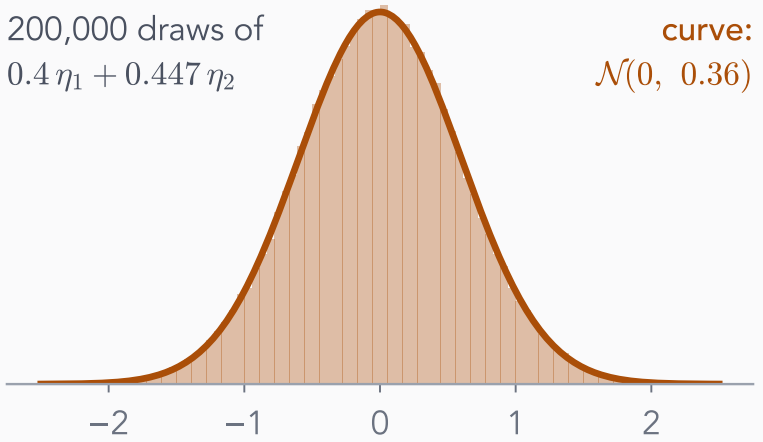
COMPUTED

Skip the loop: jump straight to step t

$$
\begin{aligned}
x_1 &= \sqrt{\alpha_1}\, x_0 + \sqrt{\beta_1}\, \eta_1 \\
x_2 &= \sqrt{\alpha_2}\, x_1 + \sqrt{\beta_2}\, \eta_2 \\
&= \sqrt{\alpha_1 \alpha_2}\, x_0 + \sqrt{\alpha_2 \beta_1}\, \eta_1 + \sqrt{\beta_2}\, \eta_2
\end{aligned}
$$

two independent noises; their variances add: $\alpha_2 \beta_1 + \beta_2 = 0.16 + 0.20 = 0.36 = 1 - \bar{\alpha}_2$

$$
x_2 = 0.8\, x_0 + 0.6\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 1)
$$



The sum of the two scaled noises is one Gaussian with variance 0.36.

$$
x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I)
$$

Any step t in one line: no loop. This is the formula training uses.

OUR PIXEL AT STEP 2

COMPUTED

Signal and noise can cancel

$0.8 \times (+1)$ signal: our ink pixel + $0.6 \times (-0.5)$ noise: $\epsilon = -0.5$ = **0.5** noisy pixel x_2

Check against the loop: with $\eta_1 = -0.5$ and $\eta_2 = -0.224$, two single steps give $x_1 = 0.671$ and then $x_2 = 0.5$. Same number.

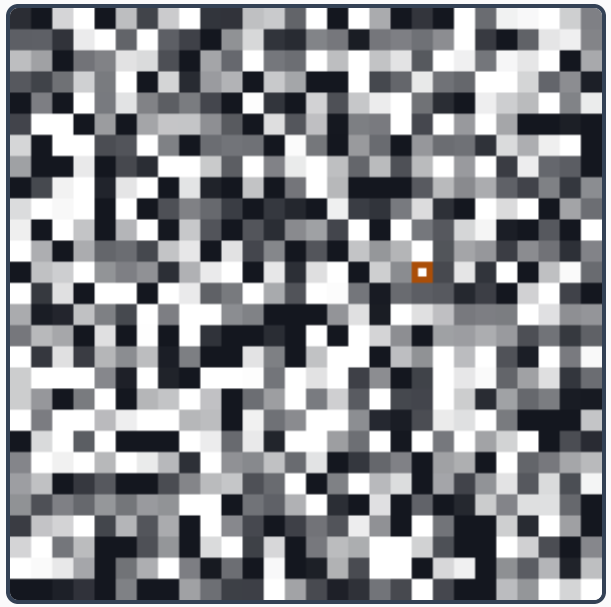
$0.8 \times (-1)$ signal: a paper pixel + $0.6 \times (+2.17)$ a large noise draw = **0.5** the same x_2

> **From 0.5 alone you cannot tell whether the pixel was ink or paper. Undoing noise needs more than this one number.**

TRY IT

COMPUTED

Slide t and watch our 7



t 920

signal scale $\sqrt{\bar{\alpha}_t}$	0.014
noise scale $\sqrt{1-\bar{\alpha}_t}$	1.000
our pixel's noisy value	−2.318
signal-to-noise $\bar{\alpha}_t/(1-\bar{\alpha}_t)$	1.9e-4

Same noise image ϵ at every t (at our pixel, $\epsilon = -2.33$).

The same formula covers every noise level: at t = 920 the image is 99.99% noise by variance.

THE FORWARD PROCESS IN CODE

Everything in Part 1 fits in a few lines of PyTorch

```
beta = torch.cat([torch.zeros(1),        # beta[0] = 0
                  torch.linspace(1e-4, 0.02, 1000)])
alpha = 1 - beta
alpha_bar = torch.cumprod(alpha, 0)      # [0] = 1

def corrupt(x0, t):       # x0 [B,1,28,28], t [B]
    eps = torch.randn_like(x0)
    ab = alpha_bar[t].reshape(-1, 1, 1, 1)
    return ab.sqrt() * x0 + (1 - ab).sqrt() * eps, eps
```

the schedule: how much noise each step adds

running product $\bar{\alpha}_t$; index t matches the maths

a fresh noise image the shape of the batch

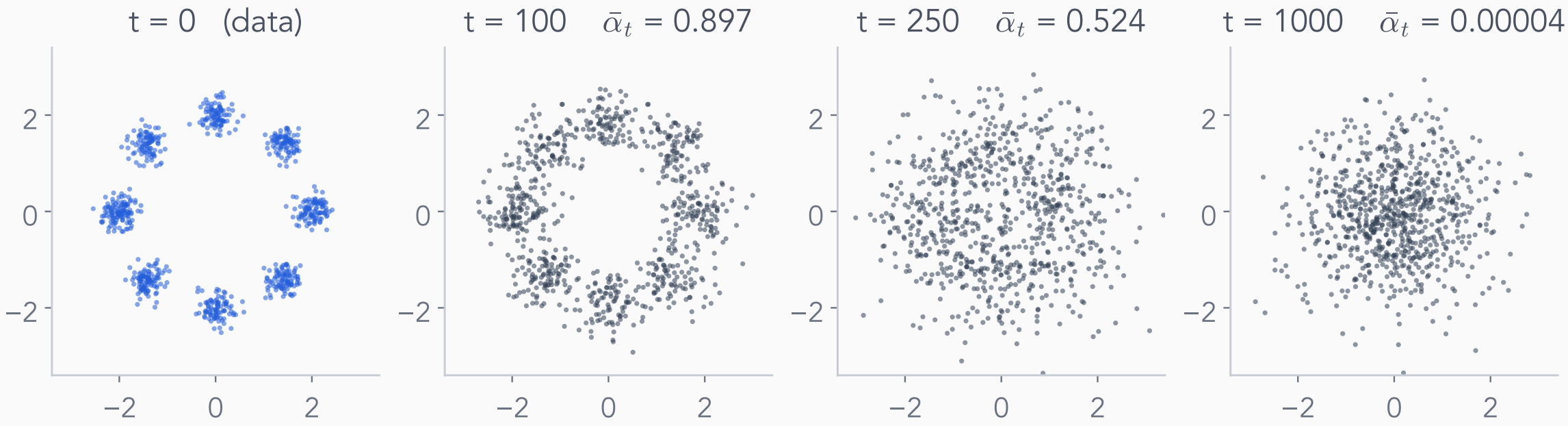
the jump formula: no loop over steps

No learning anywhere in this code. It returns the noisy input and the noise; Part 2 uses both.

THE VIEW FROM A WHOLE DATASET

COMPUTED 800 points, 2-D

Noise turns any dataset into the same Gaussian

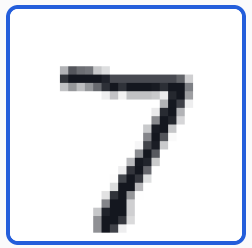


Eight tight clusters at t = 0 become one round blob, $\mathcal{N}(0, I)$, by t = 1000. Generation will start from that blob.

CAN WE UNDO IT?

COMPUTED

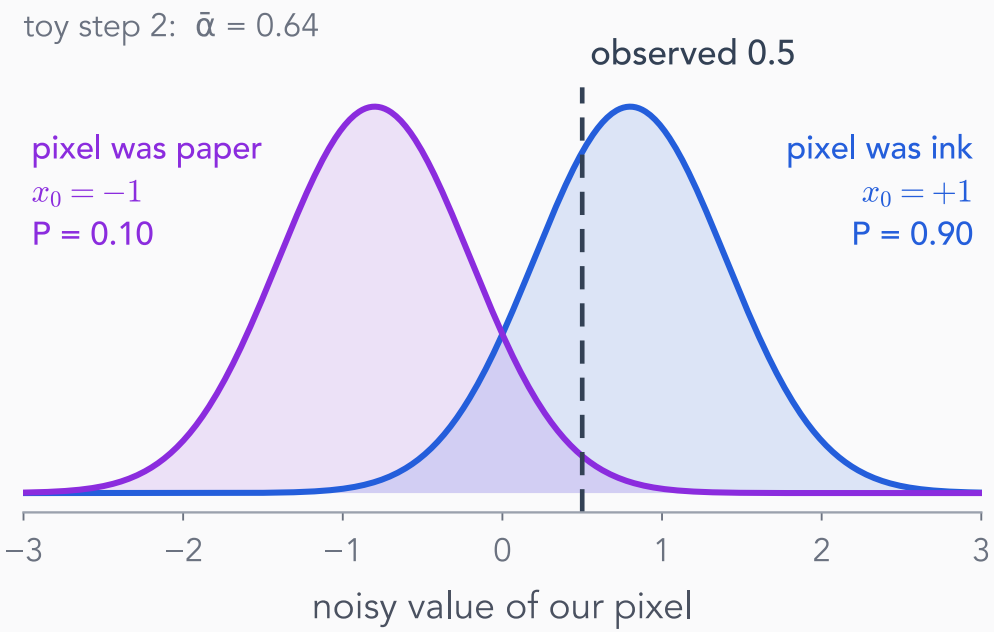
A 7 has ink at our pixel; a 2 has paper

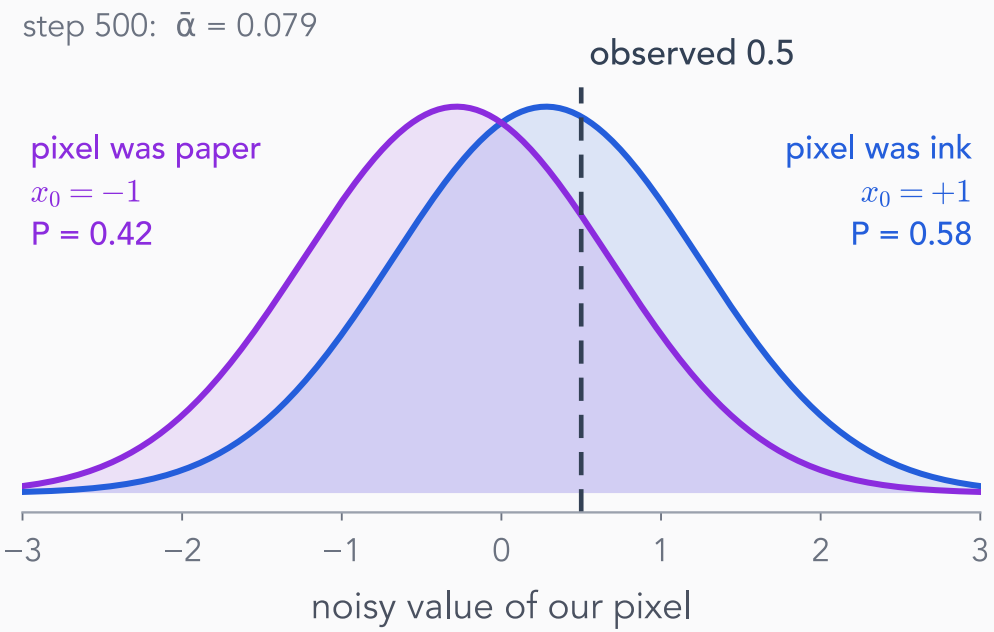


7: pixel = +1



2: pixel = −1



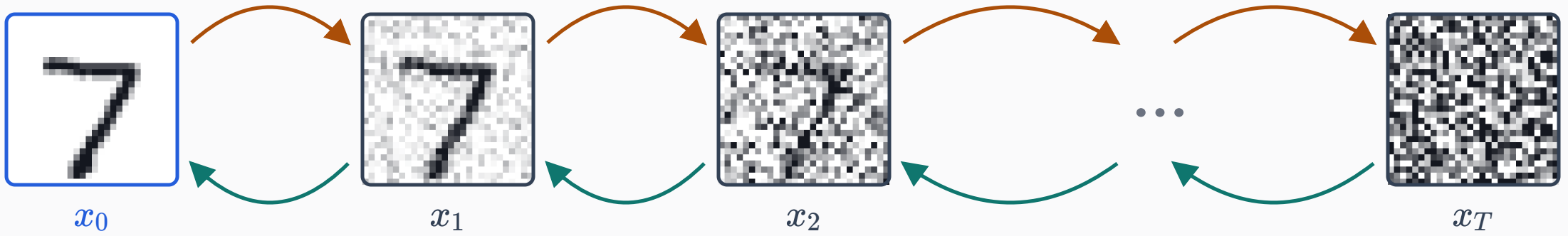


At step 2 an observed 0.5 says “ink” with probability 0.90; at step 500 only 0.58. Going backwards needs knowledge of digits, so it must be learned.

ANSWER TO QUESTION 1, AND THE SETUP FOR QUESTION 2

Forward is a fixed recipe; reverse is a model we must learn

forward $q(x_t \mid x_{t-1})$: a fixed recipe, no parameters



reverse $p_\theta(x_{t-1} \mid x_t)$: a network we must learn

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(\sqrt{1-\beta_t}\, x_{t-1},\ \beta_t I\right) \qquad p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(\mu_\theta(x_t, t),\ \sigma_t^2 I\right)$$

> Each reverse step only has to undo a small step, which is why a Gaussian with a learned mean is a good model of it.

END OF PART 1

What we did, and what comes next

WHAT WE DID

1. One step: shrink by $\sqrt{1-\beta_t}$, add $\sqrt{\beta_t}$ fresh noise; the square roots keep the variance at 1.
2. $\bar{\alpha}_t$ is the signal left after t steps: 1 at the start, 0.00004 at t = 1000.
3. Jump to any step in one line: $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon$. Our pixel: 0.5.
4. One noisy value cannot tell ink from paper, so going back needs a learned model.

NEXT: TEACH A NETWORK TO GO BACK

We can make as many noisy images as we like, and we know the noise in each.

Part 2 asks a network to predict that noise. The noise we drew becomes the label.

PART 2 OF 7

Training: a regression problem we build ourselves

Part 1 gave us a way to make noisy images. **Goal of this part:** turn that into (input, label) pairs and a loss, so that a network learns to **predict the noise**, and see what it can and cannot learn.

✓ Corrupt a digit

2 **Train: predict the noise**

3 Sample: run it backwards

4 Why this loss: the ELBO

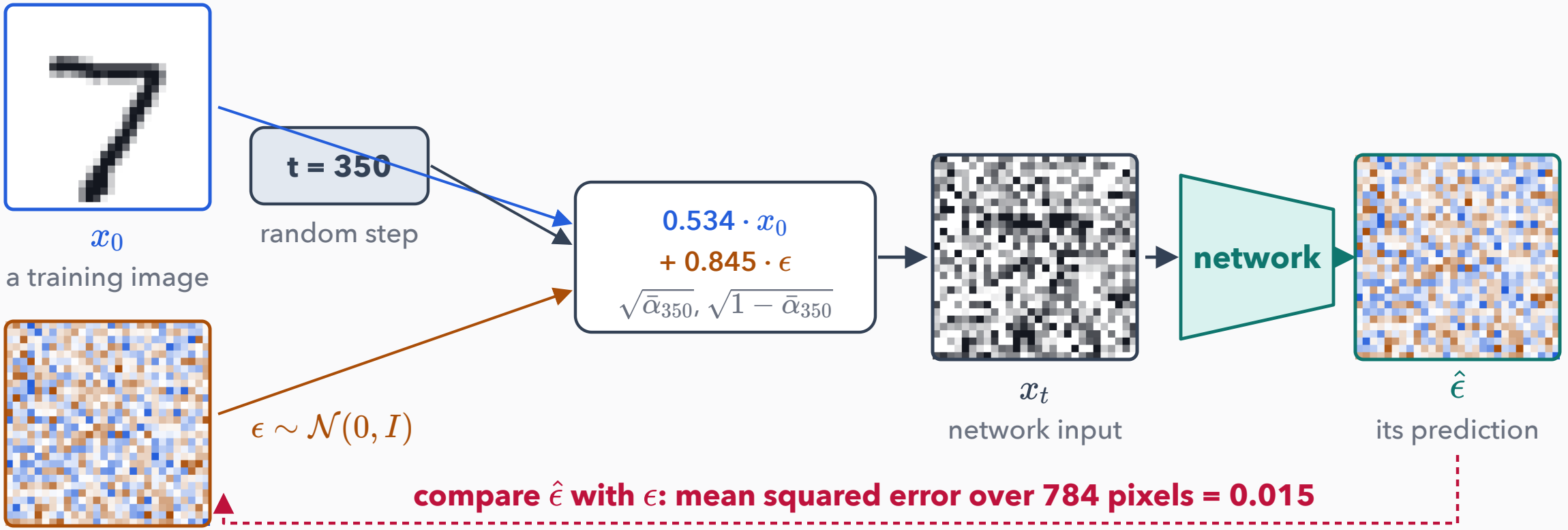
5 Inside the network

6 Steer with a condition

7 Text by denoising

OUR MODEL prediction

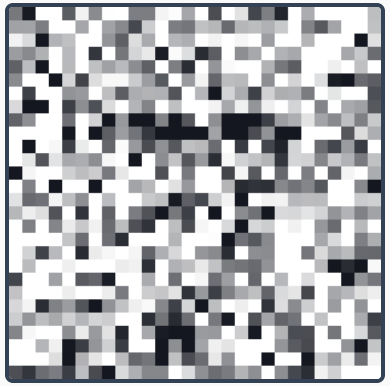
Make an input and its label; ask the network for the noise



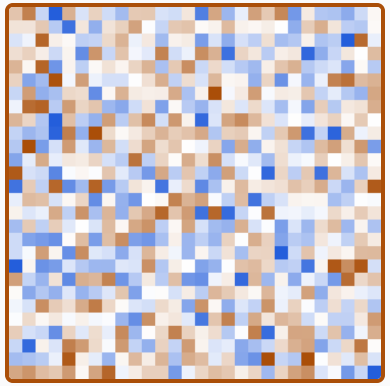
Everything except $\hat{\epsilon}$ was made by us. The noise we drew is the label: we get inputs and labels for free.

COMPUTED

From one noisy input we could ask for ϵ, for x_0, or for a mix

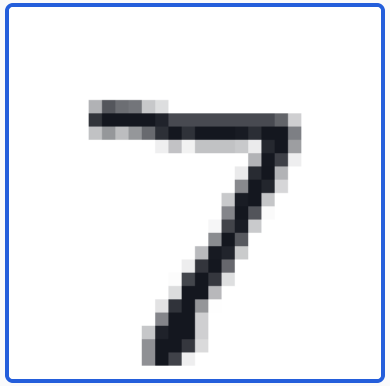


input x_t, t = 350

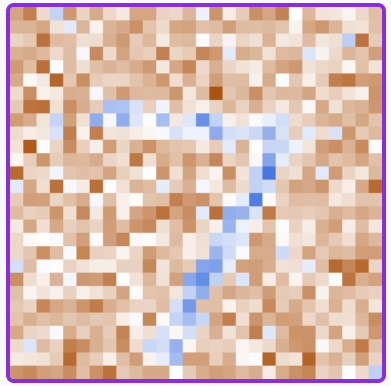


predict the noise ϵ

DDPM (Ho et al. 2020)



predict the clean image x_0



predict $v = \sqrt{\bar{\alpha}_t}\,\epsilon - \sqrt{1-\bar{\alpha}_t}\,x_0$

used by many later models

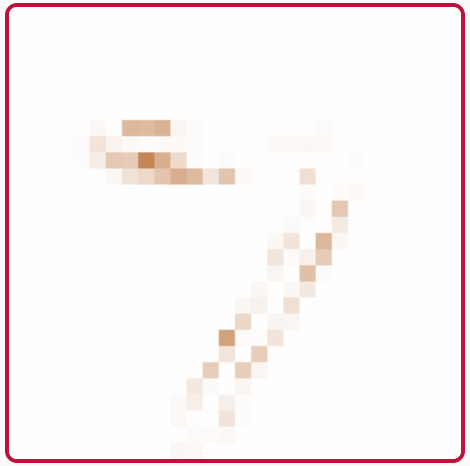
Since $x_t = \sqrt{\bar{\alpha}_t}\,x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon$, knowing x_t and any one of them gives the others.

> **We follow DDPM and predict ϵ: whatever t is, the target is always a standard Gaussian image, so the regression stays well scaled.**

OUR MODEL

Squared error, pixel by pixel, then averaged

PIXEL (ROW, COL)	TARGET E	PREDICTION E	$(E - E)^2$
ours (12,19)	−2.33	−2.34	0.000
(8,10)	1.64	1.82	0.032
(20,15)	−0.18	−0.30	0.017
(3,3)	−0.58	−0.58	0.000
mean over all 784 pixels			**0.015**



squared error at every pixel: the largest errors sit on the stroke of the 7

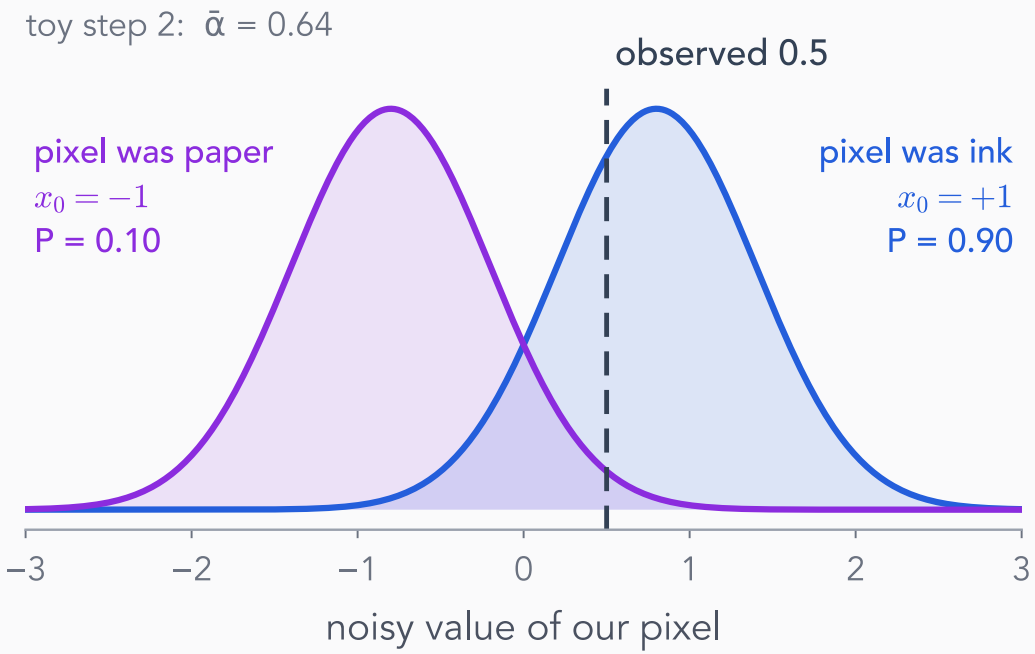
```
loss = F.mse_loss(eps_hat, eps)
```

That is the whole objective: an ordinary regression on 784 numbers.

BACK TO OUR TOY PIXEL, $X_2 = 0.5$

COMPUTED

Can the network recover the exact noise? No, so it averages



IF THE PIXEL WAS	THEN E WAS	PROBABILITY*
ink, $x_0 = +1$	−0.5	0.90
paper, $x_0 = -1$	+2.17	0.10

$$\hat{\epsilon}^\star = \mathbb{E}[\epsilon \mid x_2] = 0.90(-0.5) + 0.10(2.17) = \mathbf{-0.24}$$

Squared error is smallest for the **average** of everything that could have happened, not for the true −0.5.

* if ink and paper were equally likely before we looked

$\mathbb{E}[x_0 \mid x_2] = 0.90(+1) + 0.10(-1) = 0.80$: a grey pixel.

This is why regression blurs (the SR3 slide): it returns an average. Diffusion uses that average to take one small step, not as the final image.

THE SAME EFFECT, IN OUR TRAINED NETWORK

OUR MODEL

What the network believes the clean 7 is, at each noise level



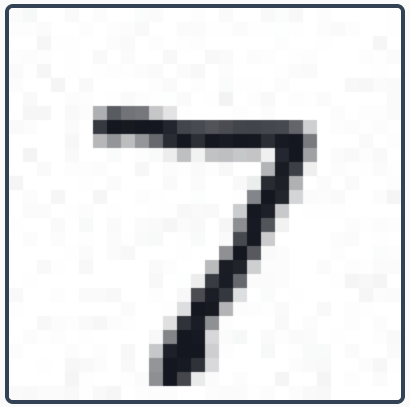
guess = the jump formula solved for x_0, with the predicted noise: $\hat{x}_0 = (x_t - \sqrt{1-\bar{\alpha}_t}\,\hat{\epsilon})/\sqrt{\bar{\alpha}_t}$

At t = 920 the guess is a blurry average of digits; by t = 350 it is our 7. The average sharpens as evidence accumulates.

ONE NETWORK, EVERY NOISE LEVEL

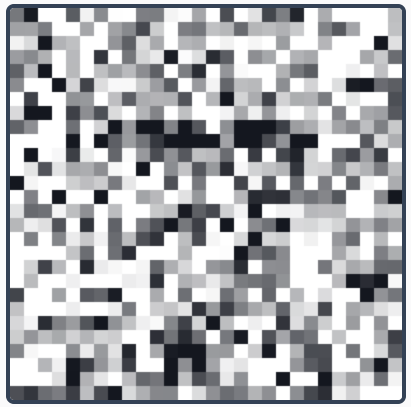
COMPUTED inputs

One network must handle nearly clean and nearly pure noise



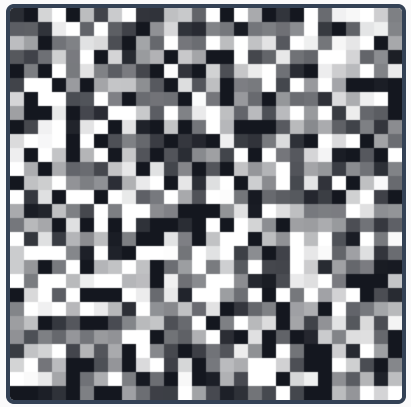
t = 7

$\bar{\alpha}_t$ = 0.999: remove a whisper



t = 350

$\bar{\alpha}_t$ = 0.285: mostly noise



t = 920

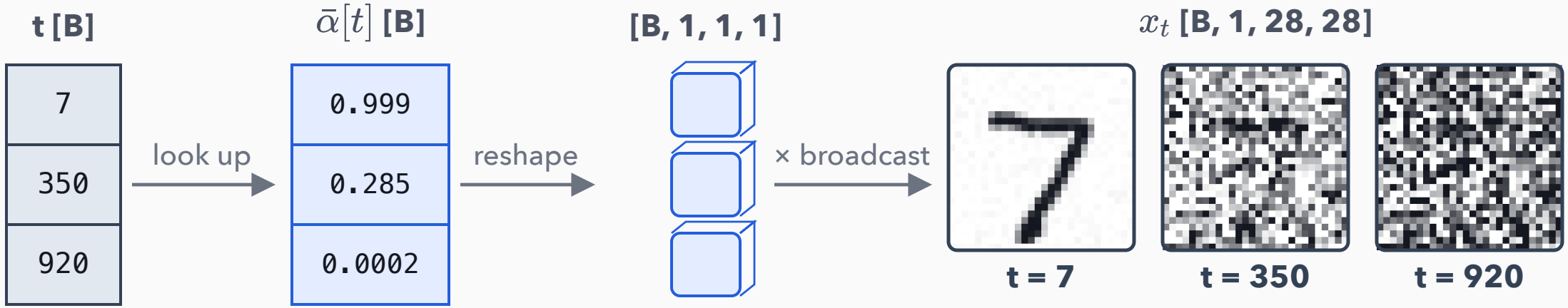
$\bar{\alpha}_t$ = 0.00019: almost all noise

How much noise to remove depends on the level, so **t is a second input**: the network computes $\hat{\epsilon} = \epsilon_\theta(x_t, t)$.

One model for all 1000 levels, told which level it is at. Part 5 shows how t enters the network.

COMPUTED

A batch mixes noise levels: one $\bar{\alpha}$ per image



each image gets its own noise level

```
ab = alpha_bar[t].reshape(B, 1, 1, 1)        # [B] -> [B, 1, 1, 1]
xt = ab.sqrt() * x0 + (1 - ab).sqrt() * eps   # [B, 1, 28, 28]
```

Random t per image means every batch trains the network at many noise levels at once.

THE COMPLETE TRAINING STEP

The whole training step in PyTorch

```
beta = torch.cat([torch.zeros(1), torch.linspace(1e-4, 0.02, 1000)])
alpha_bar = torch.cumprod(1 - beta, 0)     # once, as in Part 1

t   = torch.randint(1, T + 1, (B,))        # a step per image
eps = torch.randn_like(x0)                 # the label
ab  = alpha_bar[t].reshape(B, 1, 1, 1)
xt  = ab.sqrt() * x0 + (1 - ab).sqrt() * eps
eps_hat = model(xt, t)
loss = F.mse_loss(eps_hat, eps)
opt.zero_grad(); loss.backward(); opt.step()
```

the $\bar{\alpha}$ table; t runs from 1 to 1000

draw t and ϵ

build the noisy input

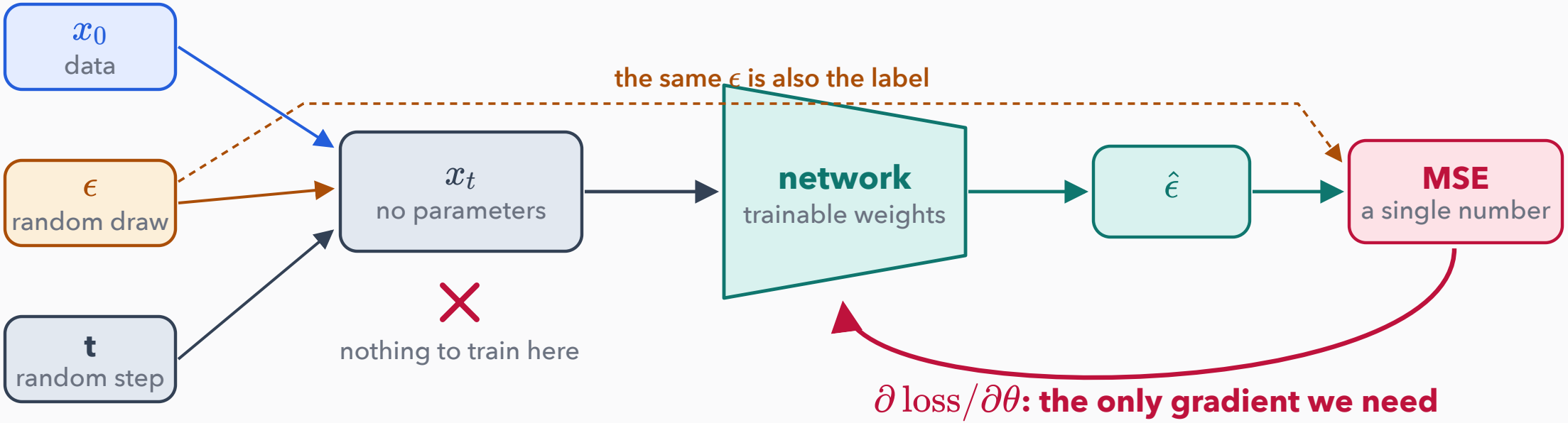
predict the noise

regress, update θ

No reverse chain, no sampling loop: one random step per image, one forward pass, one MSE.

THE COMPUTATION GRAPH

Gradients flow only into the network's weights

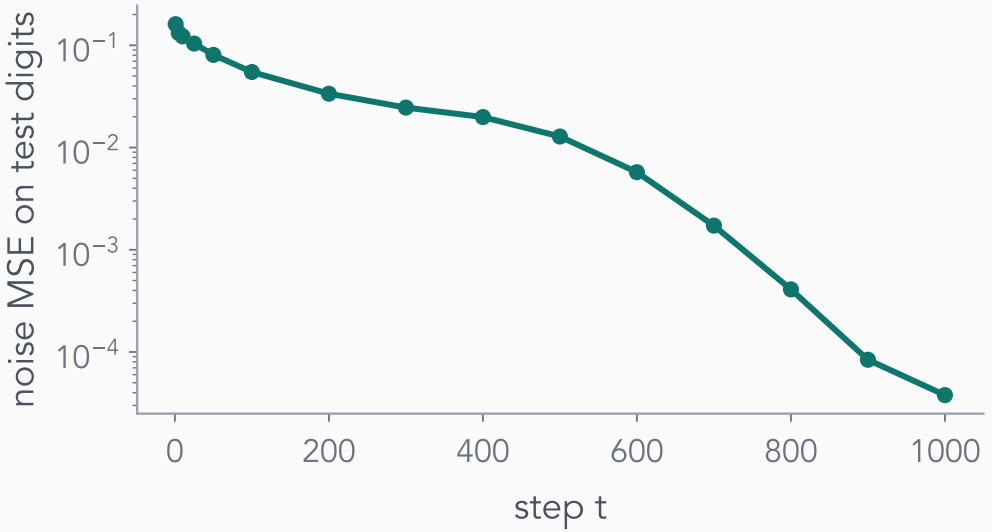


x_0, t and ϵ are constants made before the forward pass; only θ gets a gradient.

Our training run, and where the network still struggles



noise MSE during training (smoothed)



noise MSE on 512 test digits, by step t

814,145 parameters · batch 128 · 30000 steps (about 64 passes over MNIST) · about 50 minutes on an Apple-silicon GPU

> **Predicting the noise is hardest at small t (0.16 at t = 1) and easy near pure noise (0.000038 at t = 1000), where the input is almost exactly ϵ.**

END OF PART 2

What we did, and what comes next

WHAT WE DID

1. A training example is (noisy image, step t) as input and the noise ϵ as the label.
2. The loss is plain squared error over 784 pixels; one random t per image, no loop.
3. Even a perfect network cannot recover ϵ exactly; it predicts the average of what is possible: −0.24 for our pixel.
4. Our network: 814,145 parameters, 30,000 steps, hardest at small t.

NEXT: USE THE PREDICTIONS TO GENERATE

At generation time there is no clean image and no true noise, only the network.

Part 3 turns its noise prediction into one small backward step, and repeats.

PART 3 OF 7

Sampling: run the chain backwards with the network

We have a trained noise predictor. **Goal of this part:** start from pure noise and use the network to take one small backward step at a time, until a digit appears. Same toy pixel, same numbers.

✓ Corrupt a digit

✓ Train: predict the noise

3 **Sample: run it backwards**

4 Why this loss: the ELBO

5 Inside the network

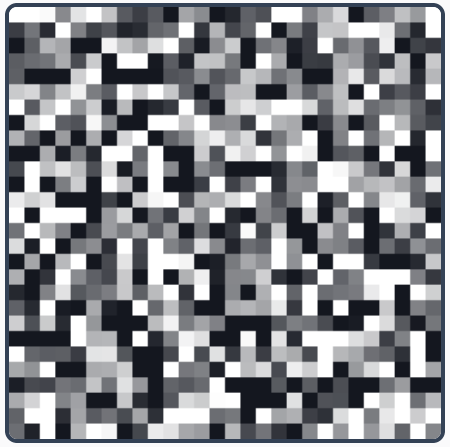
6 Steer with a condition

7 Text by denoising

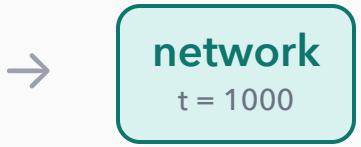
THE FIRST STEP OF GENERATION

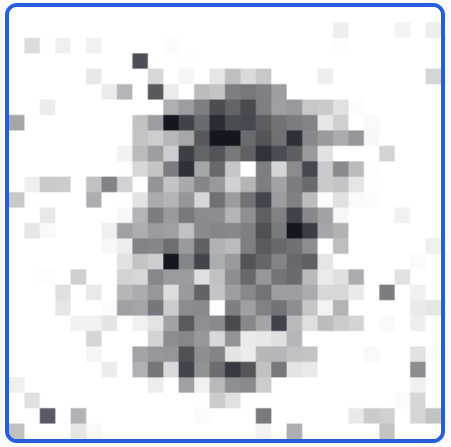
OUR MODEL

Start from noise; all we have is the network's guess



$x_{1000} \sim \mathcal{N}(0, I)$





its guess $\hat{x}_0$ of a clean image

> **The first guess is a featureless average. The trick is to move only a little way towards it, then ask again.**

OUR TOY PIXEL AGAIN: $x_2 = 0.5$, AND SUPPOSE $\hat{\epsilon} = -0.4$

COMPUTED

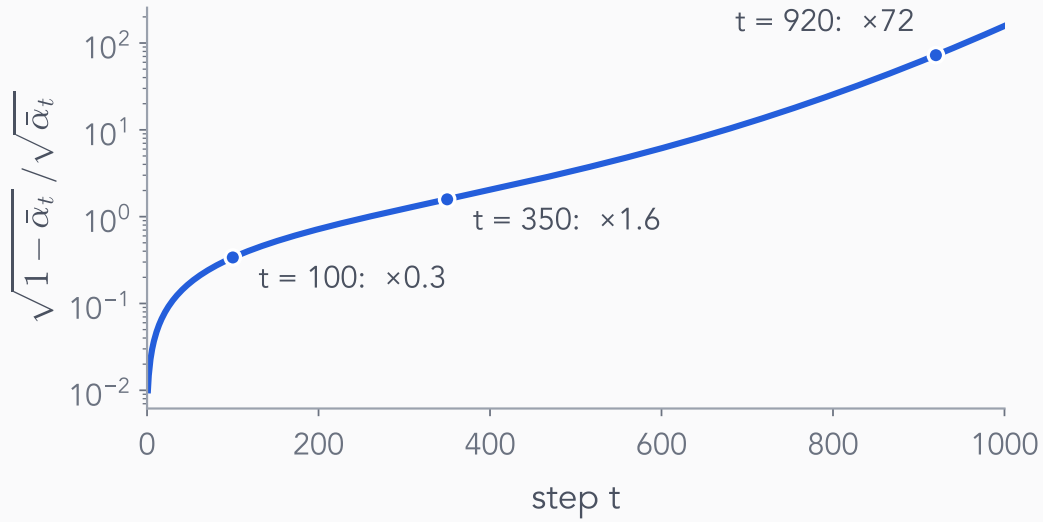
Invert the jump formula to guess the clean value

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$$

$$\Rightarrow \quad \hat{x}_0 = \frac{x_t - \sqrt{1 - \bar{\alpha}_t}\, \hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}$$

$0.5 - 0.6 \times (-0.4)$ = 0.74 $\div\ 0.8 =$ **0.925** guess of x_0

The true value was 1. A noise error of 0.1 became an error of **0.075** = (0.6 / 0.8) × 0.1.



error in $\hat{x}_0$ per unit error in $\hat{\epsilon}$, real schedule

At high noise a small error in $\hat{\epsilon}$ is magnified up to 72 times, which is why early guesses are poor and we take small steps.

IF WE KNEW THE CLEAN PIXEL

COMPUTED

Knowing x_0, the step back from x_2 is a Gaussian we can write down

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(\tilde{\mu}_t,\ \tilde{\beta}_t\right)$$

$$\tilde{\mu}_t = a_t\, x_0 + b_t\, x_t$$

$$a_t = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t},\quad b_t = \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t},\quad \tilde{\beta}_t = \beta_t\,\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$$

0.497×1 ($a_2\, x_0$: the true clean pixel) + 0.497×0.5 ($b_2\, x_2$: the noisy pixel) = **0.745** ($\tilde{\mu}_2$)

Variance $\tilde{\beta}_2 = 0.2 \times 0.2/0.36 = 0.111$. A blend of where we are and where the clean image is, plus some spread.

> **This is the "true" reverse step. It needs x_0, which we have during training but not when generating.**

WE DO NOT KNOW x_0, SO USE THE GUESS

COMPUTED

The model's reverse step: the same blend, with $\hat{x}_0$ in place of x_0

0.497×0.925	$+$	0.497×0.5	$=$	**0.708**
$a_2\,\hat{x}_0$: the guess		$b_2\,x_2$		μ_θ, the model's mean

$$\mu_\theta(x_t, t) = a_t\,\hat{x}_0 + b_t\,x_t$$

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\left(\mu_\theta,\ \tilde{\beta}_t\right)$$

True mean 0.745, model mean 0.708: they differ by 0.037 only because $\hat{\epsilon} = -0.4$ instead of -0.5. Part 4 turns this gap into the loss.

> **A better noise prediction gives a better clean guess, which gives a reverse step closer to the true one.**

ONE REVERSE STEP · PART 1

COMPUTED

The same mean, written with the predicted noise

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\, \hat{\epsilon} \right)$$

0.5 (x2) − (0.2 / 0.6) × (−0.4) ($c_2\hat{\epsilon} = -0.133$) = 0.633 $\div\sqrt{0.8}$ = **0.708** (μ_θ)

Substitute $\hat{x}_0 = (x_t - \sqrt{1-\bar{\alpha}_t}\,\hat{\epsilon})/\sqrt{\bar{\alpha}_t}$ into the blend and simplify: the same 0.708, computed directly from $\hat{\epsilon}$.

Read it as: remove a fraction $\beta_t/\sqrt{1-\bar{\alpha}_t}$ of the predicted noise, then rescale by $1/\sqrt{\alpha_t}$. The code uses this form.

ONE REVERSE STEP · PART 2

COMPUTED

Then add fresh noise, except at the very last step

$$x_{t-1} = \mu_\theta + \sigma_t\, z, \qquad z \sim \mathcal{N}(0, I)$$

$$\sigma_t^2 = \tilde{\beta}_t = \beta_t \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}$$

$$\sigma_2^2 = 0.2 \times \frac{0.2}{0.36} = 0.111$$

With $z = 0.3$: $x_1 = 0.708 + 0.333 \times 0.3 =$ **0.808**

At t = 1 we return the mean with no noise: the output is the network's final estimate.

$\hat{\epsilon}$	−0.400
z	0.300

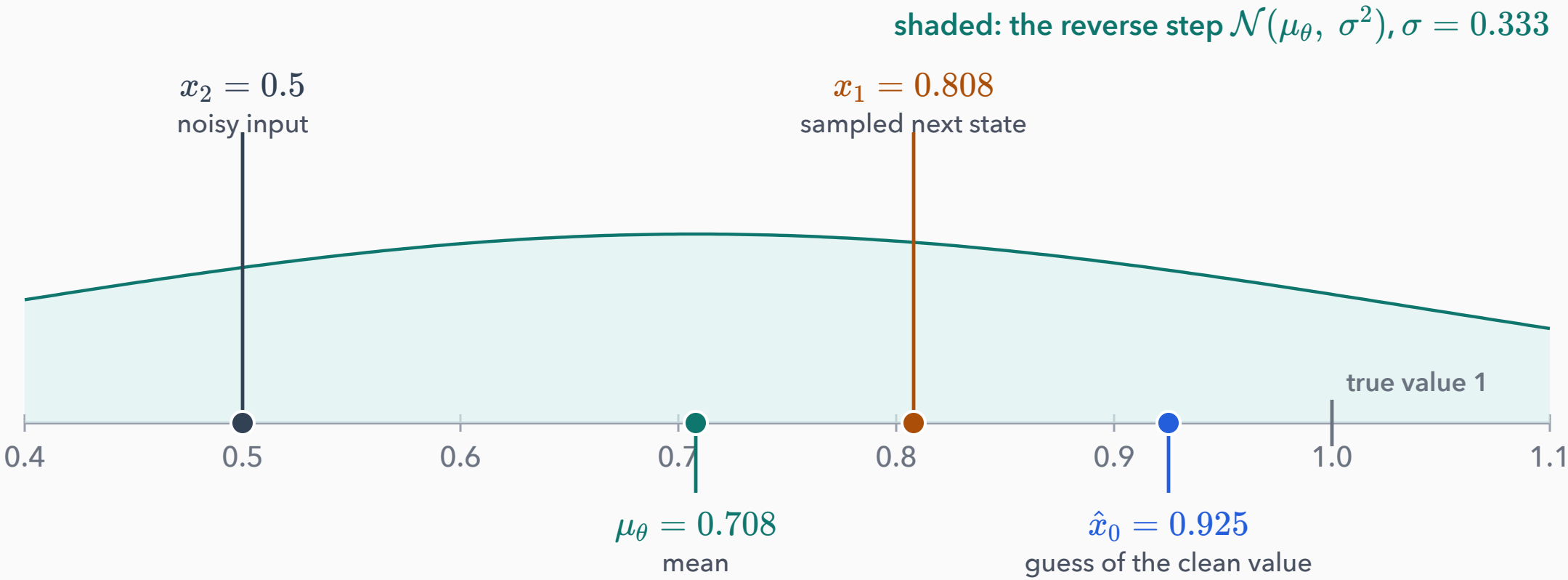
mean μ_θ	0.708
fresh noise σz	0.100
next state x_1	0.808
clean guess $\hat{x}_0$	0.925

> **The mean says where to go; the noise keeps each step a sample from a distribution rather than a single best guess.**

DO NOT CONFUSE THESE

COMPUTED

Four numbers on one line



The sampler keeps $x_1 = 0.808$ and feeds it back in. The guess $\hat{x}_0 = 0.925$ is only for looking at.

OUR MODEL

Run our model with and without the fresh noise

normal sampler
mean + $\sigma_t z$



mean only
same starting noise,
$\sigma_t = 0$



Average distance between the 8 images: 21.9 with noise, 0.6 without.

Without fresh noise, all eight starting points end on the same thin “1”. Following only the mean leads every start to the same average answer.

THE COMPLETE SAMPLER

The loop from the first slide, in ten lines

```
x = torch.randn(B, 1, 28, 28)                # x_T
for t in range(T, 0, -1):                    # 1000, 999, ..., 1
    tt = torch.full((B,), t)
    eps_hat = model(x, tt)
    c = beta[t] / (1 - alpha_bar[t]).sqrt()
    mean = (x - c * eps_hat) / alpha[t].sqrt()
    if t > 1:
        x = mean + post_var[t].sqrt() * torch.randn_like(x)
    else:
        x = mean                             # last step: no noise
```

start from pure noise

one network call per step, same weights

the mean formula

fresh noise with the shape of x

1000 network calls for one batch of images. Sampling is slow; training was one call per example.

WHAT WE WATCH DURING GENERATION

OUR MODEL

The state, and the network's guess at every stage

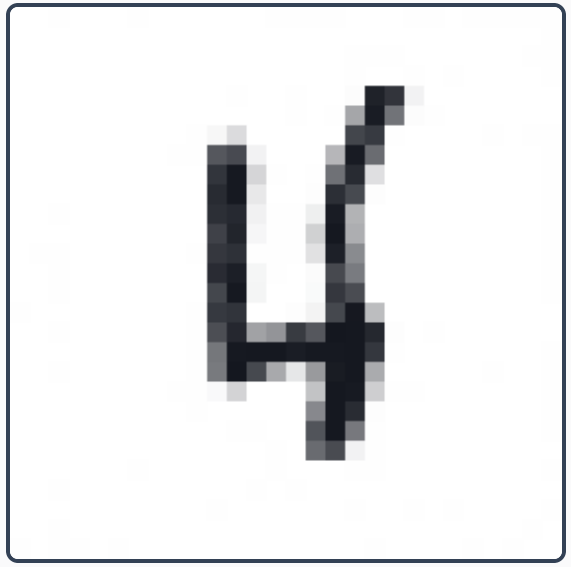


The guess commits to a digit long before the state looks clean: the overall shape is decided at high noise, details at low noise.

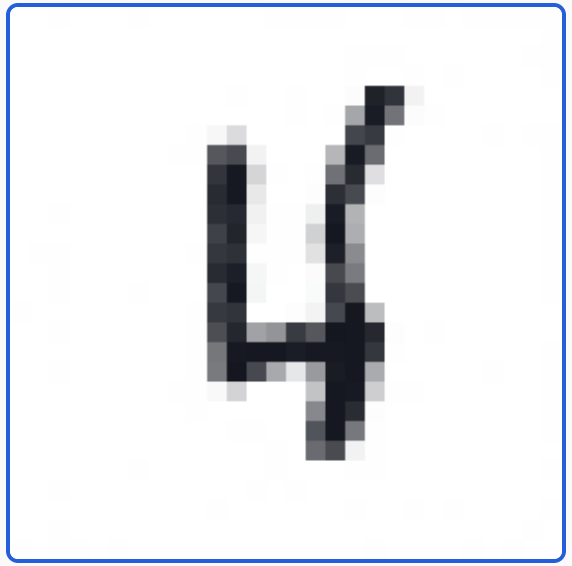
TRY IT

OUR MODEL

Scrub through a sample as it forms



state x_t



the network's guess $\hat{x}_0$

step 0

seed 1 2 3 4 5 6

Recorded from our model every 20 steps; nothing is computed in the browser.

Drag slowly through t = 600 … 300: that is where each seed decides which digit it will be.

EVIDENCE, NOT A SHOWCASE

OUR MODEL seed 667, unselected

32 samples from our model, in the order they came out





top: our first six samples · bottom: the closest of the 60,000 training images

The samples resemble training digits without copying them. Judge a generator by an unselected batch, and check for memorisation.

END OF PART 3

What we did, and what comes next

WHAT WE DID

1. Invert the jump formula to guess the clean image: $\hat{x}_0 = 0.925$ (true 1).
2. Knowing x_0, the backward step is a Gaussian with mean 0.745; with the guess instead, 0.708.
3. Add fresh noise $\sigma_t z$ at every step but the last: 0.708 → 0.808. Without it, every sample is the same "1".
4. Repeat 1000 times from pure noise: the loop on the first slide.

NEXT: WAS "PREDICT THE NOISE" THE RIGHT LOSS?

Part 2 chose noise MSE because it was easy. The two means on this slide, 0.745 and 0.708, are exactly what the ELBO compares.

Part 4 shows the ELBO becomes a weighted noise MSE.

PART 4 OF 7

Why is noise MSE the right loss?

Part 2 chose “predict the noise” because it is easy to set up. Now that Part 3 has given us both reverse means, the true one and the model's, we can show it is what the **ELBO from Lecture I** asks for, up to a weighting.

✓ Corrupt a digit

✓ Train: predict the noise

✓ Sample: run it backwards

4 **Why this loss: the ELBO**

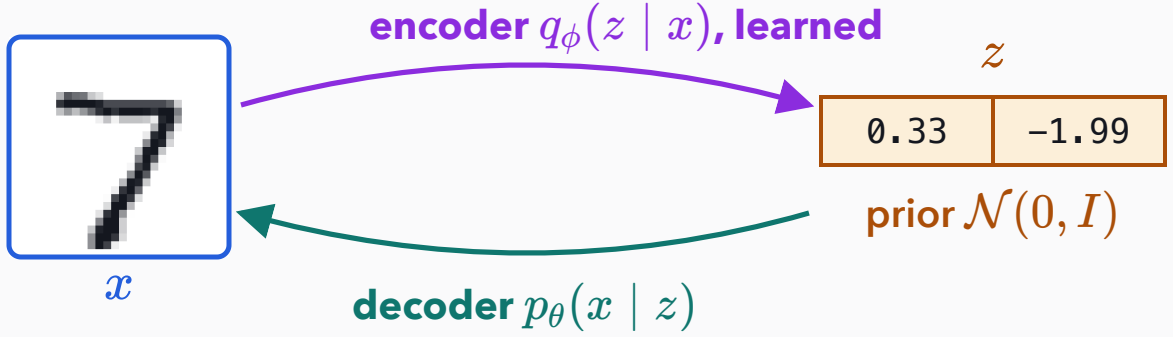
5 Inside the network

6 Steer with a condition

7 Text by denoising

A VAE has one latent; diffusion has a chain of them

Lecture I · VAE



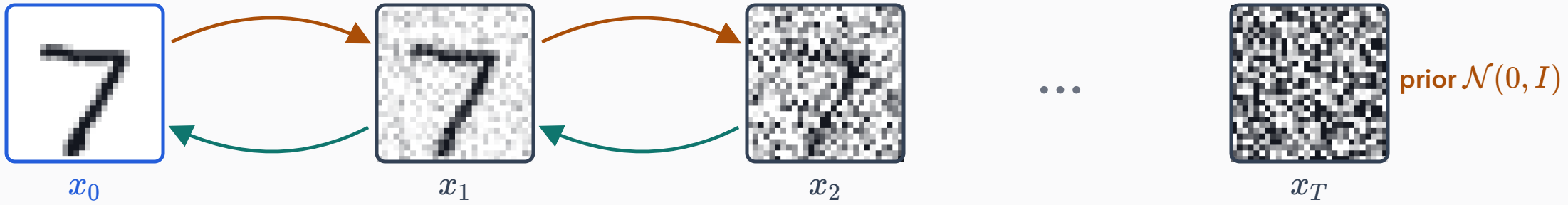
the same three ingredients

latent: $z \leftrightarrow x_1, \ldots, x_T$

decoder: one pass $\leftrightarrow p_\theta(x_{t-1} \mid x_t)$, T passes

encoder: learned ↔ fixed

Diffusion



> If diffusion is a VAE with a fixed encoder, the VAE's training objective, the ELBO, should apply to it.

LECTURE I'S ELBO, AGAIN

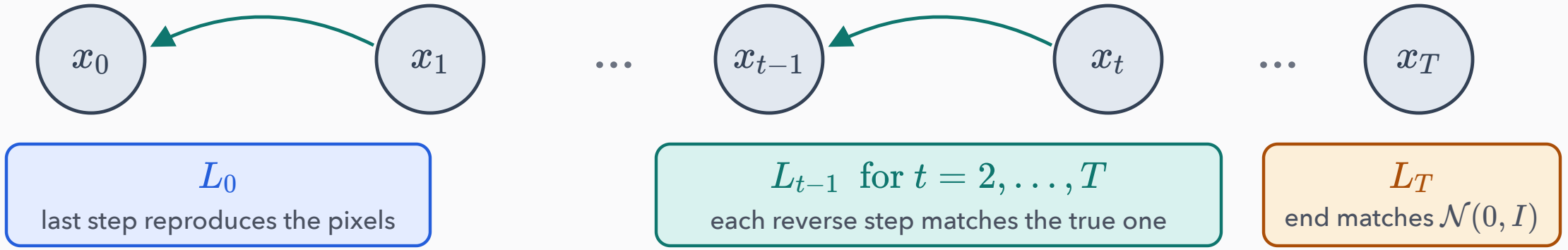
Diffusion is a VAE with a fixed encoder

	LECTURE I: VAE	DIFFUSION
latent	one code z, 2–16 numbers	$x_1, \ldots, x_T$: each the size of the image
encoder	$q_\phi(z \mid x)$, learned	$q(x_{1:T} \mid x_0)$: **fixed** noising, nothing to learn
decoder	$p_\theta(x \mid z)$, one network pass	$p_\theta(x_{t-1} \mid x_t)$, the same network used T times
prior	$p(z) = \mathcal{N}(0, I)$	$p(x_T) = \mathcal{N}(0, I)$
bound	$\log p(x) \geq$ reconstruction $-$ KL	$\log p_\theta(x_0) \geq \mathbb{E}_q\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right]$, the same Jensen step

Same ELBO derivation as Lecture I. The new part is how it splits up, one term per step.

TAKING THE BOUND APART

The bound splits into one comparison per step



$$-\text{ELBO} = L_T + \sum_{t=2}^{T} L_{t-1} + L_0$$

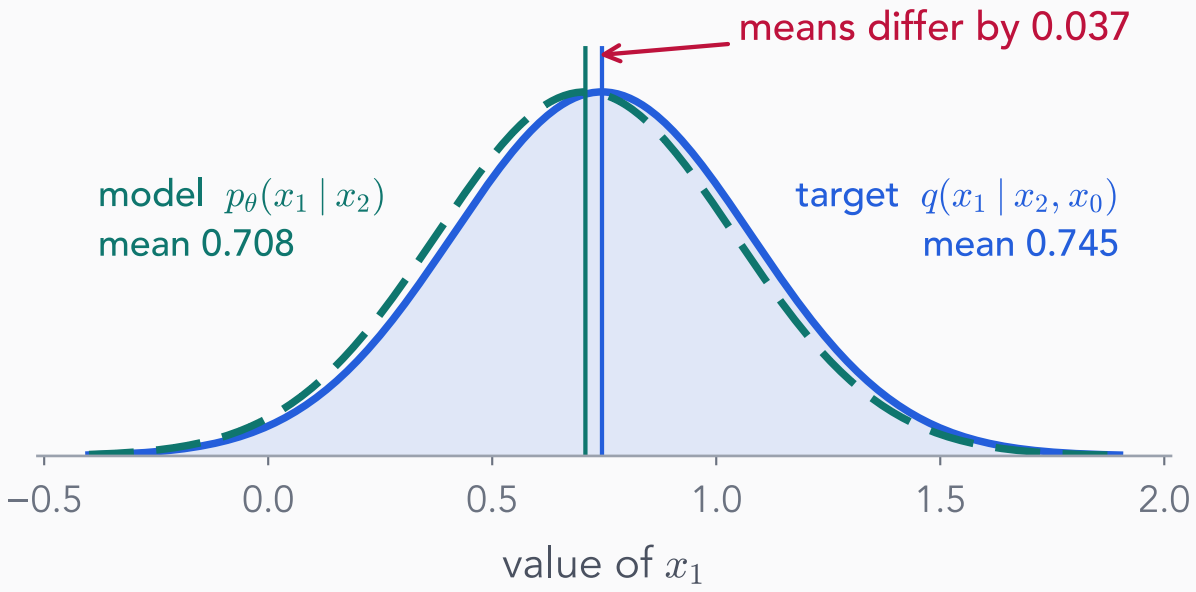
$$L_{t-1} = \mathbb{E}_q \, D_{KL}\big(q(x_{t-1} \mid x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t) \big)$$

> L_T has no parameters; L_0 is a small pixel-level term. Training is really about the middle terms.

ONE MIDDLE TERM, FOR OUR TOY PIXEL AT T = 2

COMPUTED

Equal variances: KL is the squared gap between the means



$$D_{KL} = \frac{(\tilde{\mu} - \mu_\theta)^2}{2\sigma^2}$$

true reverse mean $\tilde{\mu} = 0.745$ (we know $x_0 = 1$)
model's mean $\mu_\theta = 0.708$ (uses $\hat{\epsilon} = -0.4$)
both variances $\sigma^2 = 0.111$

$$D_{KL} = \frac{0.0373^2}{2 \times 0.111} = 0.00625$$

> **Training pushes the model's reverse mean onto the true one. Next: rewrite both means in terms of noise.**

THE ONE SUBSTITUTION THAT EXPLAINS THE LOSS

COMPUTED

Write both means with noise: the KL becomes noise MSE

model: $\mu_\theta = \frac{1}{\sqrt{\alpha_t}}\left(x_t - c_t\,\hat{\epsilon}\right)$ — $c_t = \beta_t/\sqrt{1-\bar{\alpha}_t}$, from Part 3

true: $\tilde{\mu} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - c_t\,\epsilon\right)$ — the same formula with the true noise (it is the posterior mean of Part 3)

gap: $\tilde{\mu} - \mu_\theta = \frac{c_t}{\sqrt{\alpha_t}}\left(\hat{\epsilon} - \epsilon\right)$ — x_t cancels

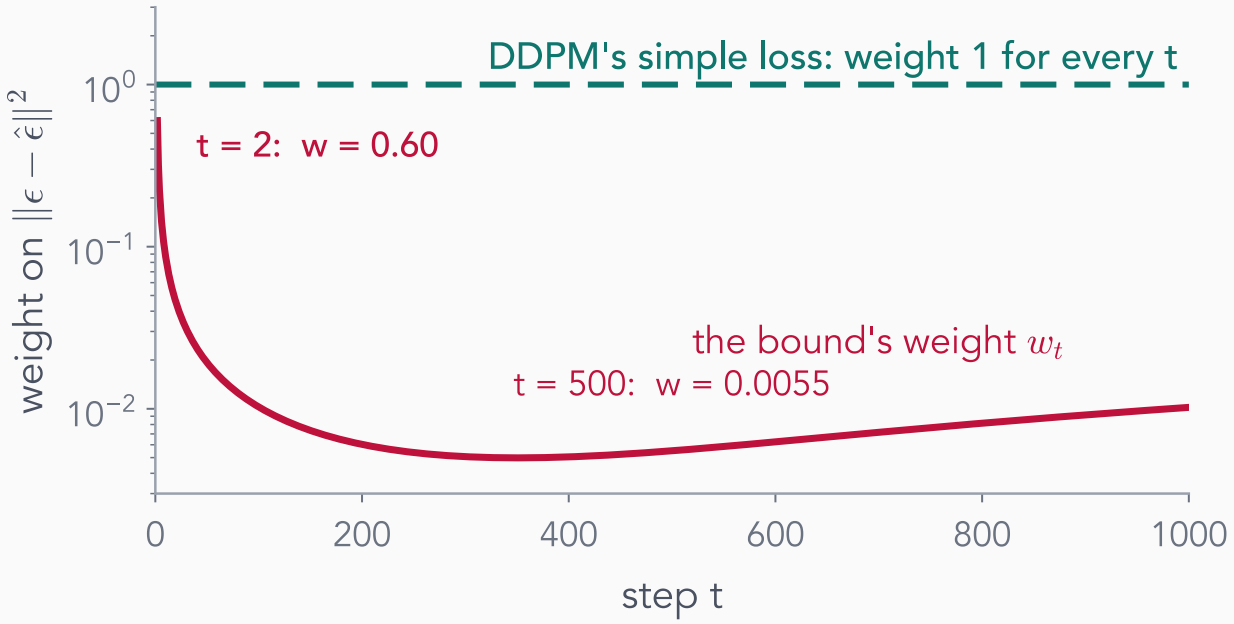
so: $D_{KL} = \underbrace{\frac{\beta_t^2}{2\sigma_t^2\,\alpha_t\,(1-\bar{\alpha}_t)}}_{w_t}\,(\hat{\epsilon}-\epsilon)^2$ — a weighted squared error on the noise

$w_2 = 0.625$ toy weight at t = 2 × $(-0.4+0.5)^2 = 0.01$ our pixel's noise error = **0.00625** the KL from the last slide

> **Each ELBO term is a weighted MSE between predicted and true noise, which is why we trained on noise.**

COMPUTED

DDPM keeps the squared error and drops the weights



$$L_{\text{simple}} = \mathbb{E}_{x_0,t,\epsilon} \left\| \epsilon - \epsilon_\theta(x_t, t) \right\|^2$$

The bound's weight falls about 109× from t = 2 to t = 500. Setting every weight to 1 gives **relatively more weight to noisy steps** than the bound does.

It also drops L_T (no parameters) and the discrete pixel term L_0.

> **The training loss is not the ELBO; it is a re-weighted sum of its per-step terms, chosen because it gave better samples (Ho et al. 2020).**

END OF PART 4

What we showed, and what comes next

WHAT WE DID

1. Diffusion is a VAE whose encoder is the fixed noising chain.
2. Its ELBO is one KL per reverse step.
3. Each KL is a weighted squared error on the noise: 0.625×0.1^2 for our pixel.
4. DDPM trains with weight 1 at every step instead.

NEXT: THE NETWORK ITSELF

The whole method treats the network as a box that takes x_t and t and returns a noise image. Part 5 opens the box.

PART 5 OF 7

Inside the denoiser: networks you already know

So far the network was a box. **Goal of this part:** open it: a U-Net for pixels, a Transformer for patches, and why big models denoise a compressed image.

✓ Corrupt a digit

✓ Train: predict the noise

✓ Sample: run it backwards

✓ Why this loss: the ELBO

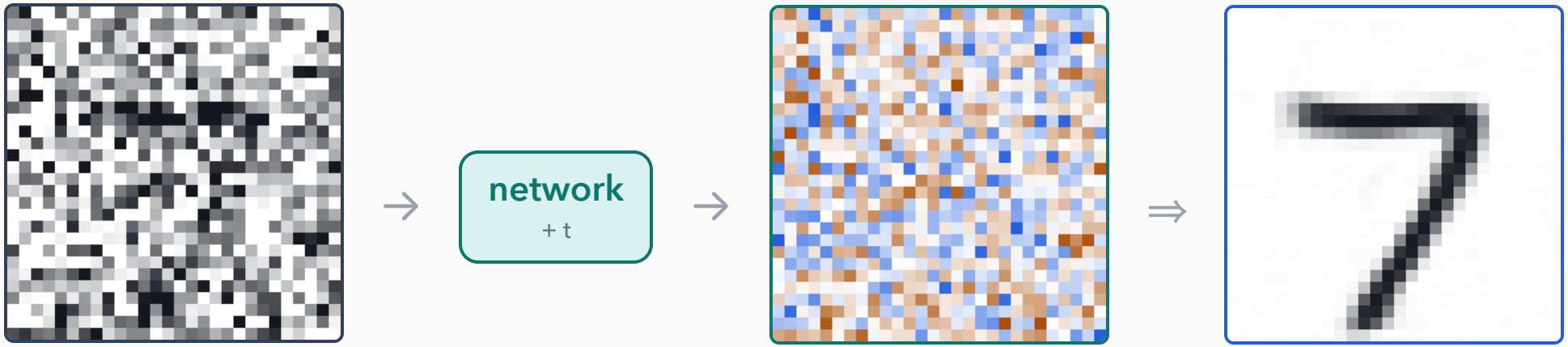
5 **Inside the network**

6 Steer with a condition

7 Text by denoising

OUR MODEL

Same shape in, same shape out: a dense prediction



x_t: [B, 1, 28, 28]

$\hat{\epsilon}$: [B, 1, 28, 28]

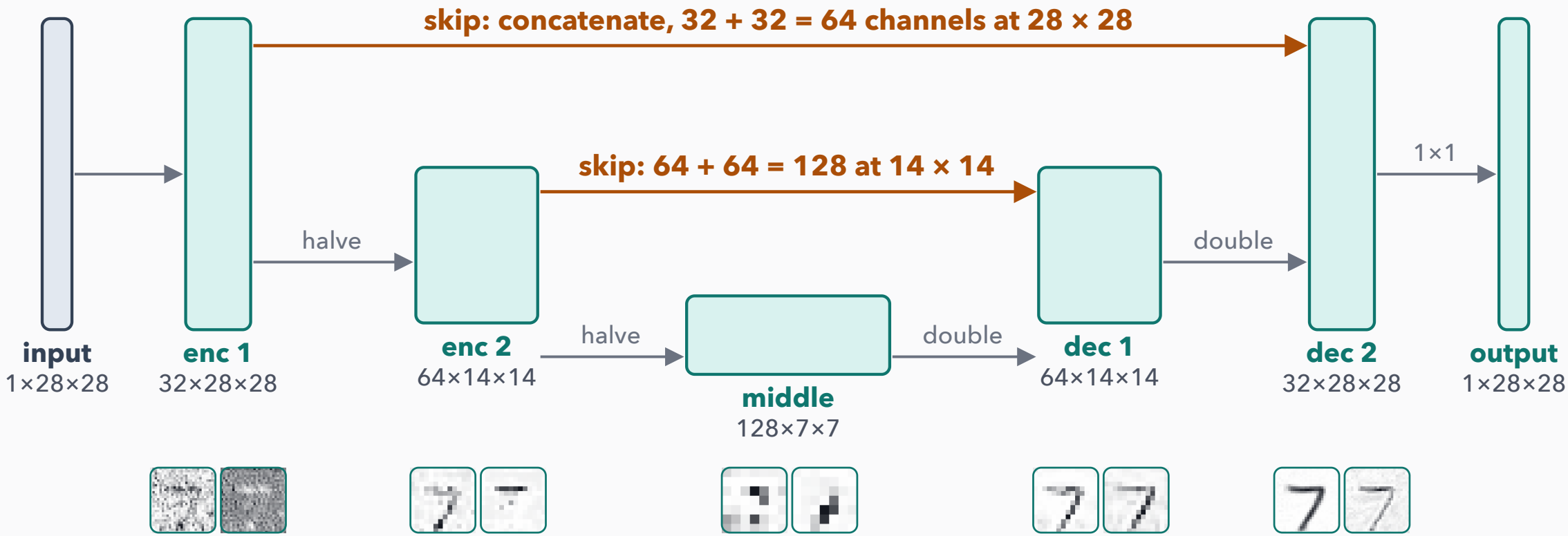
implied $\hat{x}_0$

Like segmentation, there is one output per pixel. Unlike segmentation, each output is a real number (noise), not a class.

So the architectures that work for dense prediction (encoder–decoders with skips) work here.

OUR MODEL feature maps

A small U-Net: shrink to see context, grow back for detail

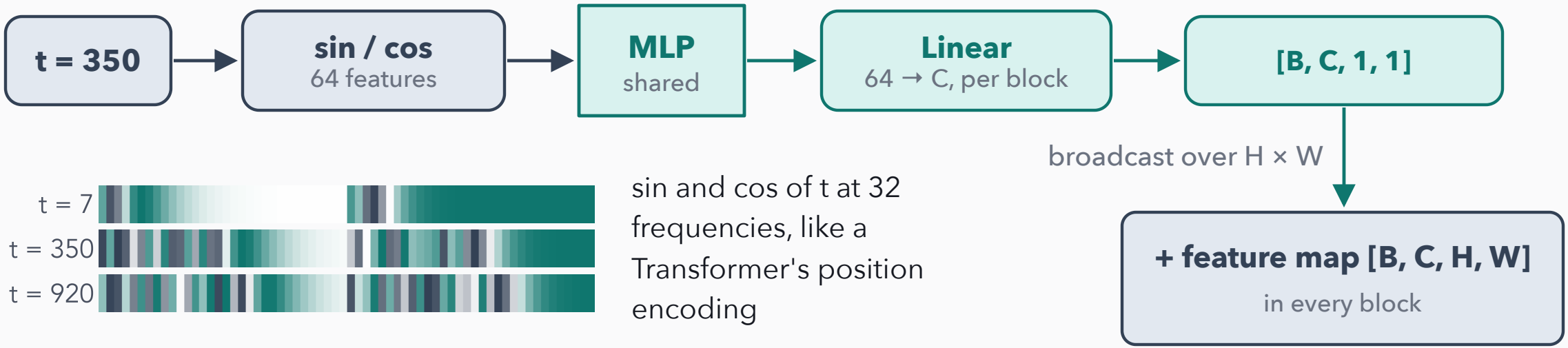


two channels per level for our noisy 7 at t = 350: the trained network's activations

> **814,145 parameters. The skips carry fine detail across; the middle sees the whole digit at 7 × 7.**

TELLING THE NETWORK WHICH STEP IT IS AT

t becomes a vector, then a per-channel offset in every block



Every block can change its behaviour with the noise level, at almost no cost in parameters.

PUBLISHED SD v1 shapes

Run 1000 steps on fewer numbers

	IMAGE	LATENT	POSITIONS	NUMBERS
Stable Diffusion v1	3 × 512 × 512	4 × 64 × 64	64× fewer	48× fewer
our MNIST autoencoder	1 × 28 × 28	4 × 7 × 7	16× fewer	4× fewer

786,432 numbers → 16,384. Positions shrink by 8 × 8 = 64; the latent keeps 4 channels instead of 3, so numbers shrink by 48.

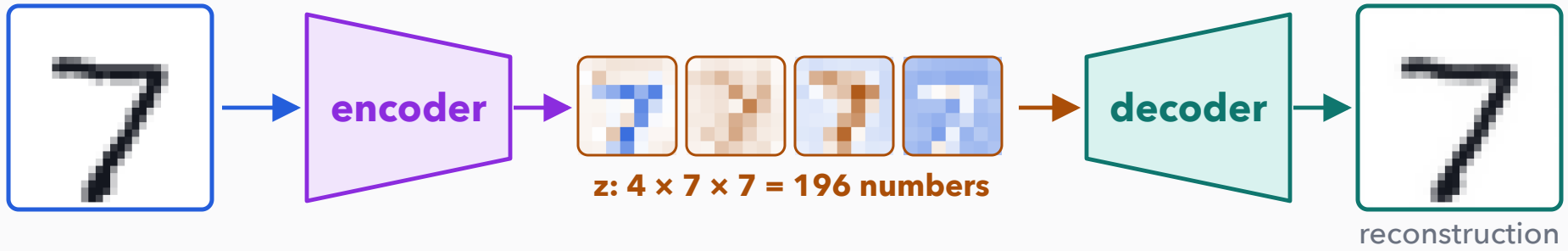
> **The latent is still spatial (a small image), so the same U-Net or DiT works on it. The autoencoder handles fine pixel detail once.**

LATENT DIFFUSION, TRAINED IN TWO STAGES

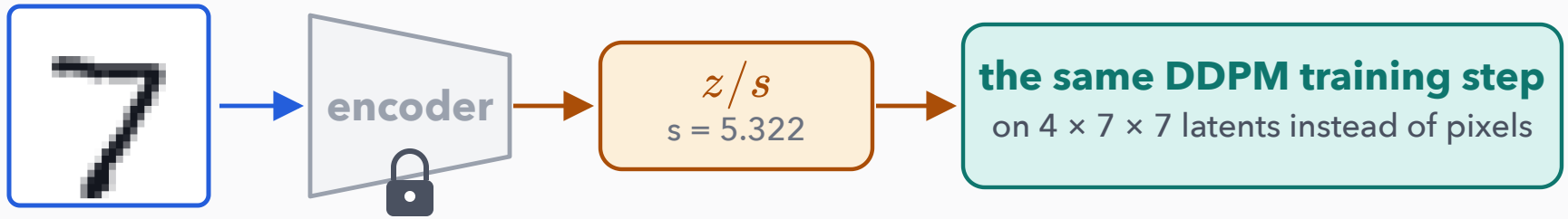
OUR MODEL

First an autoencoder, then diffusion on its latents

Stage 1 · train an autoencoder (reconstruction loss)



Stage 2 · freeze the autoencoder, train a denoiser on z

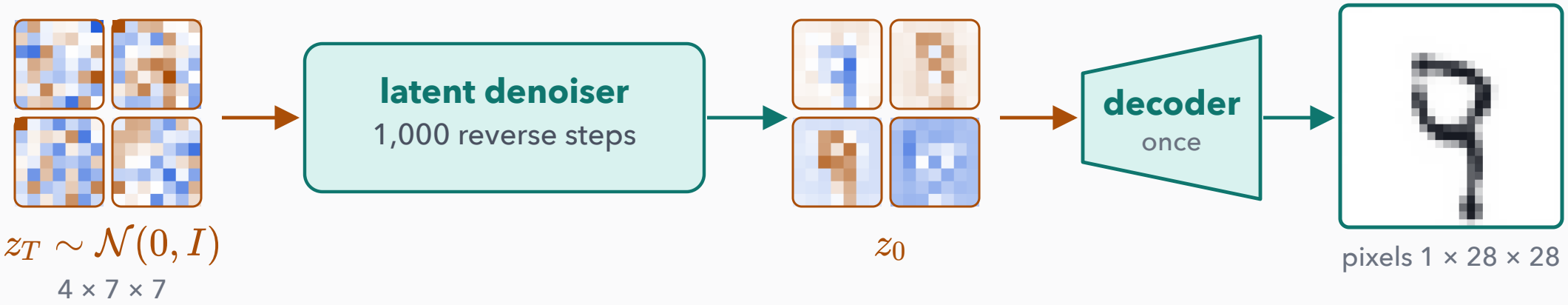


the decoder is not used in stage 2

> The diffusion target is noise in latent coordinates; nothing about the training step changes.

OUR MODEL

Denoise latents for 1000 steps, decode once

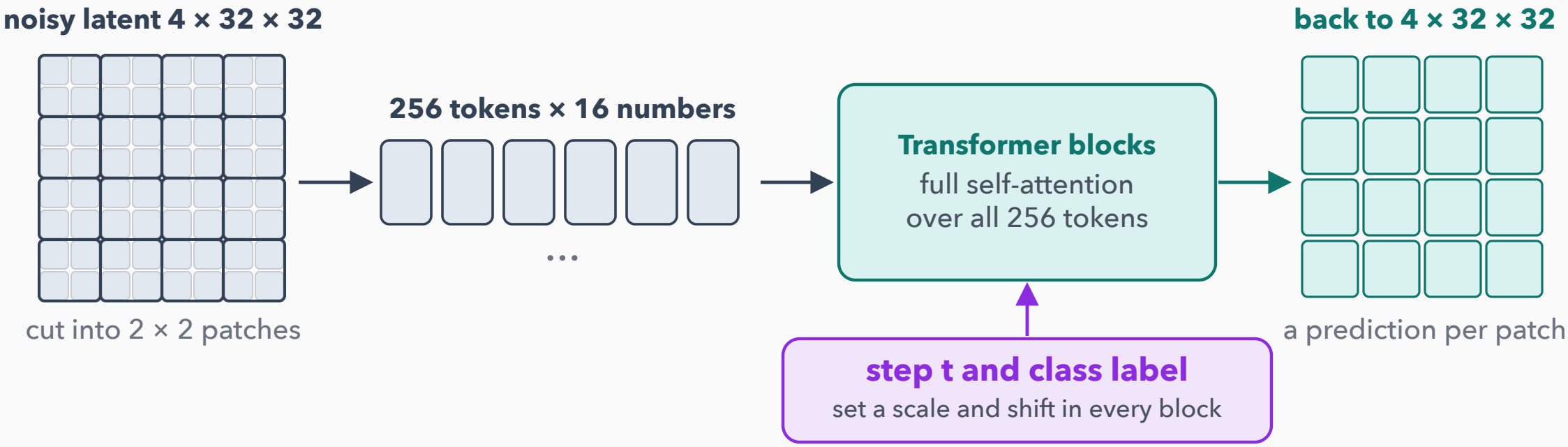




> No encoder at generation time: text-to-image starts from latent noise. Image-to-image tasks (SDEdit) can start from an encoded input instead.

PUBLISHED Peebles & Xie 2023 shapes

A Transformer can be the denoiser too: DiT



> **Patches in, a prediction for every patch out: no class token, because the output is dense. t and the label set each block's scale and shift.**

DiT: Peebles & Xie, Scalable Diffusion Models with Transformers, 2023 (class-conditional, latent space, 256 × 256 images).

PART 6 OF 7

Steering the denoiser with a prompt, a label or known pixels

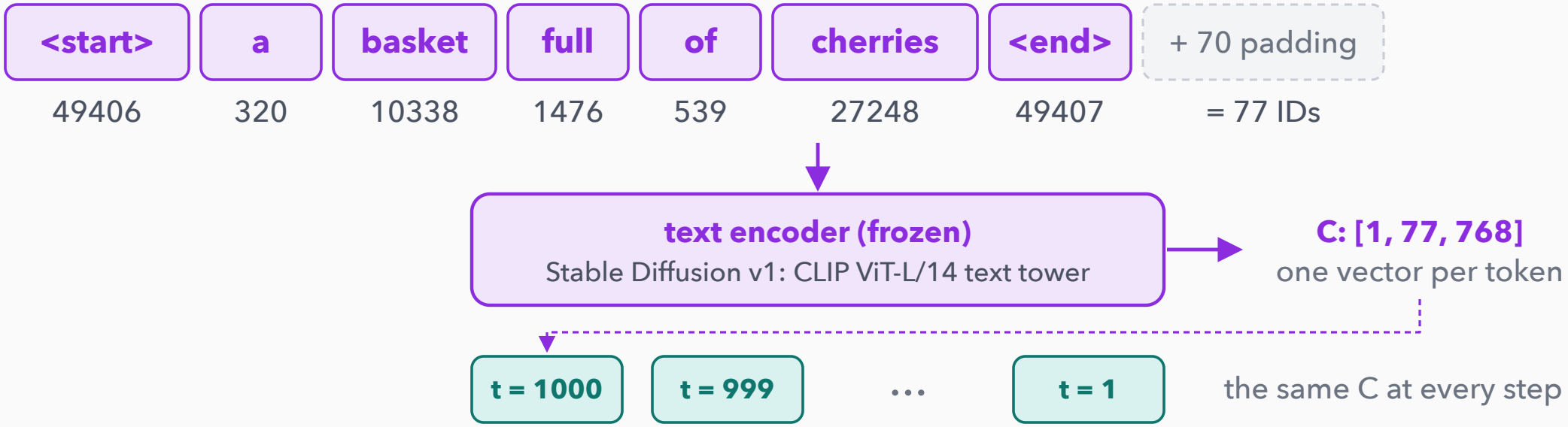
Everything so far generates a random digit. **Goal of this part:** make it generate what we ask for. Add the condition c as another input, $\epsilon_\theta(x_t, t, c)$; the training step is unchanged.

✓ Corrupt a digit
✓ Train: predict the noise
✓ Sample: run it backwards
✓ Why this loss: the ELBO
✓ Inside the network
6 **Steer with a condition**
7 Text by denoising

BACK TO “A BASKET FULL OF CHERRIES”

COMPUTED real CLIP tokenizer

What the network receives from a prompt

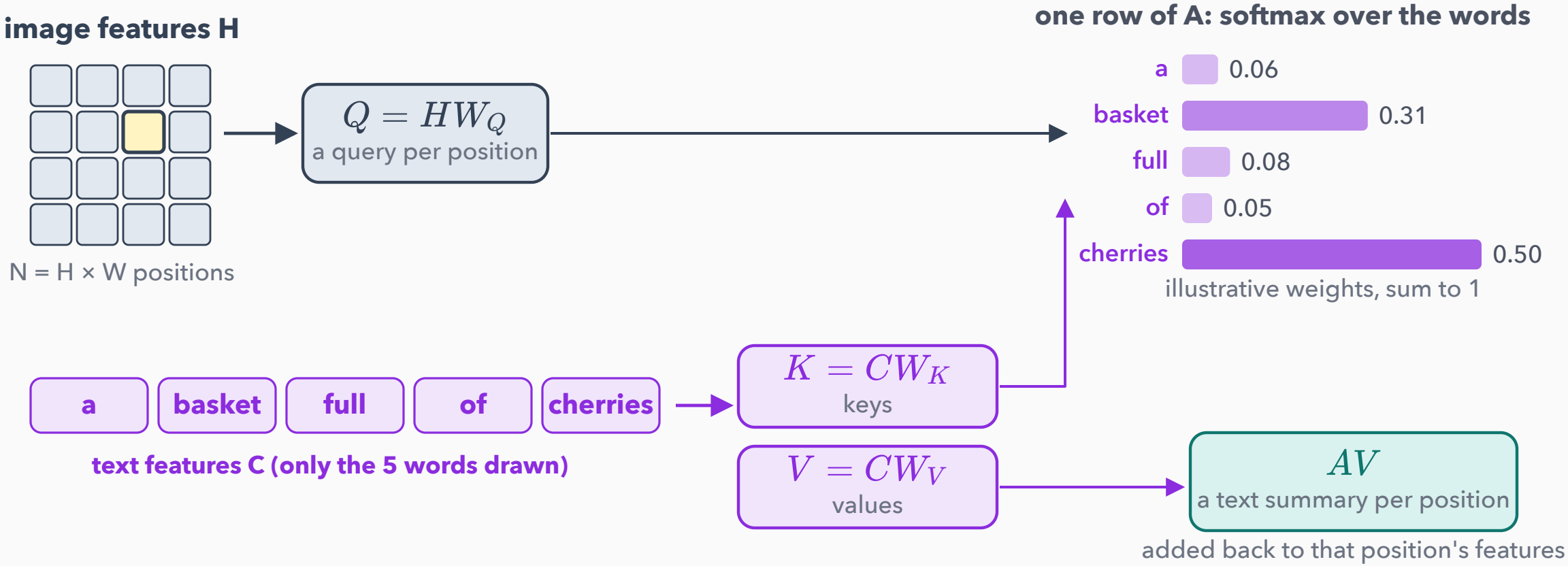


The prompt becomes 77 vectors, computed once and reused at every one of the denoising steps.

CROSS-ATTENTION, REVERSED FROM THE VLM LECTURE

ILLUSTRATIVE WEIGHTS

Image positions ask questions; words supply the answers



In a VLM, text queried image features. Here image features query the text: each position pulls in the words relevant to it.

ONE IMAGE QUERY, THREE WORDS, WORKED BY HAND

ILLUSTRATIVE hand-chosen numbers

Scores, softmax over the words, weighted sum of values

WORD	KEY K	VALUE V
golden	(1, 0)	(1, 0)
dog	(0, 1)	(0, 2)
grass	(1, 1)	(1, 1)

query 1 = (1, 0)

scores $q \cdot k / \sqrt{2}$: (0.707, 0, 0.707)

softmax: **(0.401, 0.198, 0.401)** sums to 1

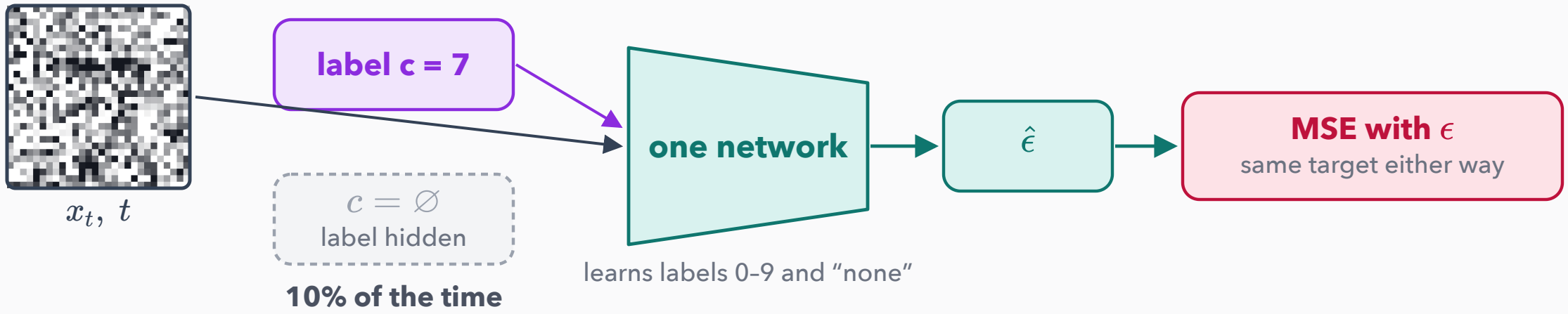
output: 0.401·(1,0) + 0.198·(0,2) + 0.401·(1,1) = **(0.802, 0.797)**

query 2 = (0, 1): softmax (0.198, 0.401, 0.401), output **(0.599, 1.203)**

> **Softmax runs over the words, separately for each image position: every position gets its own mixture of the prompt.**

OUR MODEL

Train one network with the label, and sometimes without it

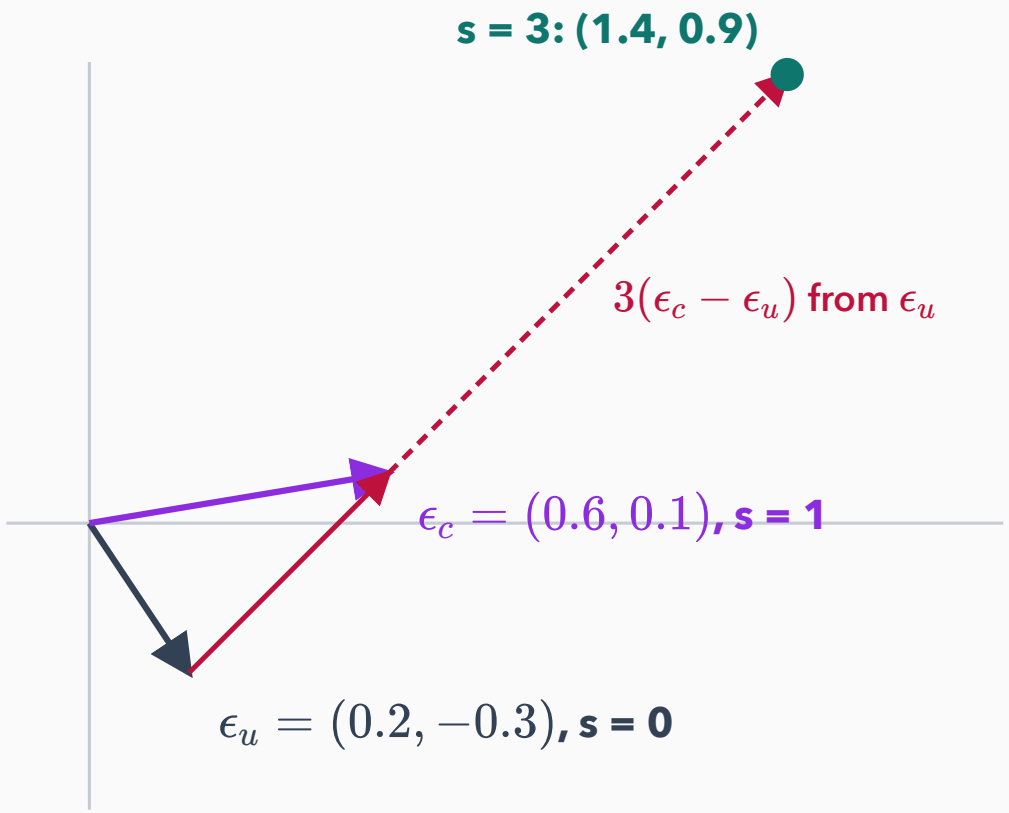



```
c = torch.where(torch.rand(B) < 0.1, torch.full_like(c, 10), c)   # 10 means "no label"
eps_hat = model(xt, t, c)
```

The same weights now give two predictions for any noisy image: ϵ_c with the label and ϵ_u without.

OUR MODEL bottom row

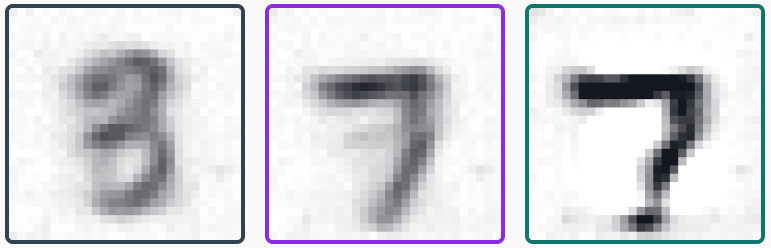
Guidance: step past the conditional prediction



illustrative 2-D vectors, equal axis scales

$$\epsilon_{\text{cfg}} = \epsilon_u + s\,(\epsilon_c - \epsilon_u)$$

$s = 0$: ignore the label · $s = 1$: ordinary conditional · $s > 1$: exaggerate the label's direction



guess, s = 0 guess, s = 1 guess, s = 3

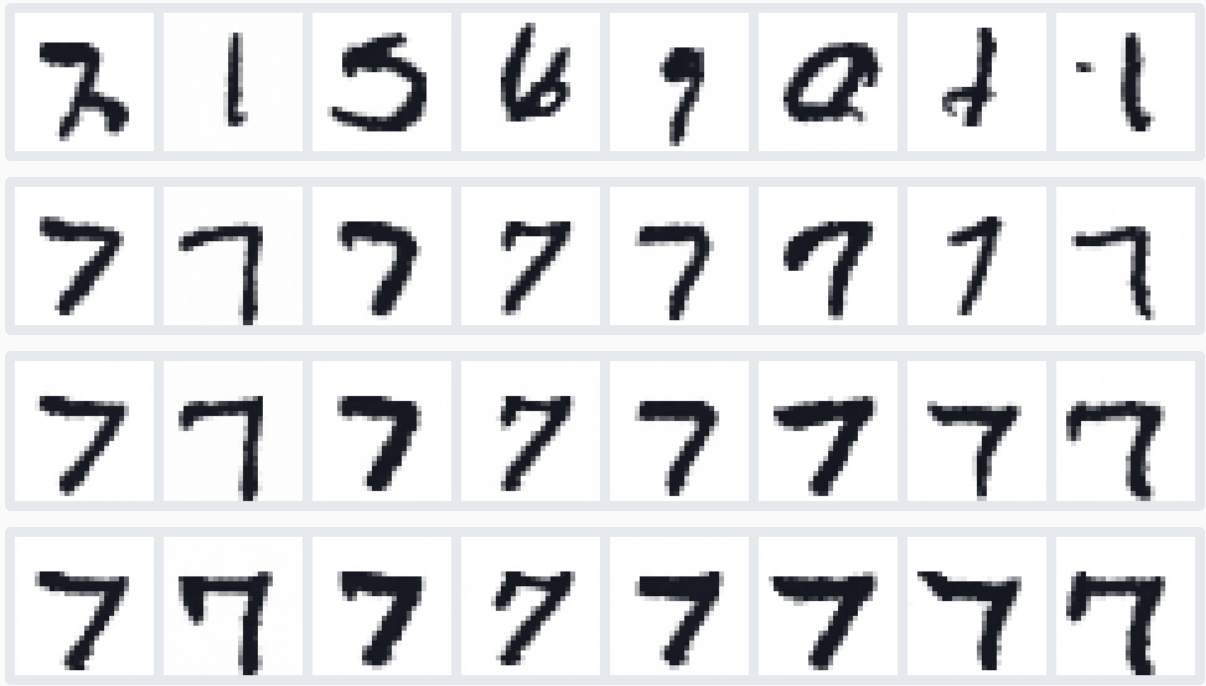
Our model, label 7, one noisy blank page at t = 600: the implied clean image for each prediction.

Two network calls per step, combined by one line of arithmetic. No separate classifier is trained.

SAME 64 STARTING NOISES, LABEL 7, FOUR GUIDANCE SCALES

OUR MODEL

Turning up guidance trades variety for fidelity

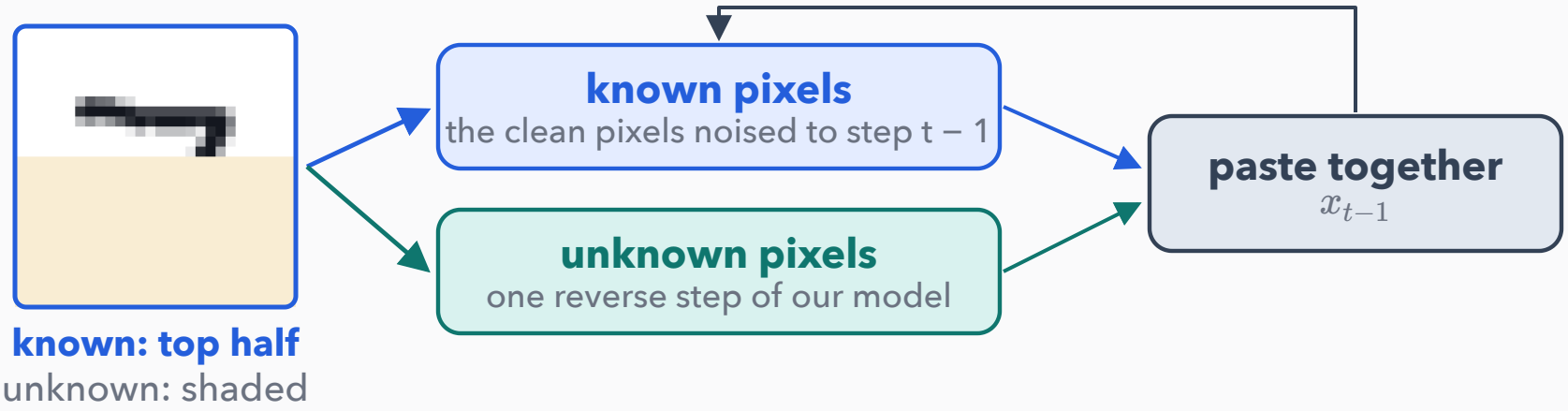


"Look like a 7": a separate MNIST classifier's verdict on all 64 samples. Spread: average pixel distance between samples. 8 of 64 shown.

> **At s = 1, 95% already look like a 7. Larger s gives 100%, with bolder and more alike strokes: spread 16.4 → 15.7.**

OUR MODEL

Inpainting with our model: paste the known pixels back in



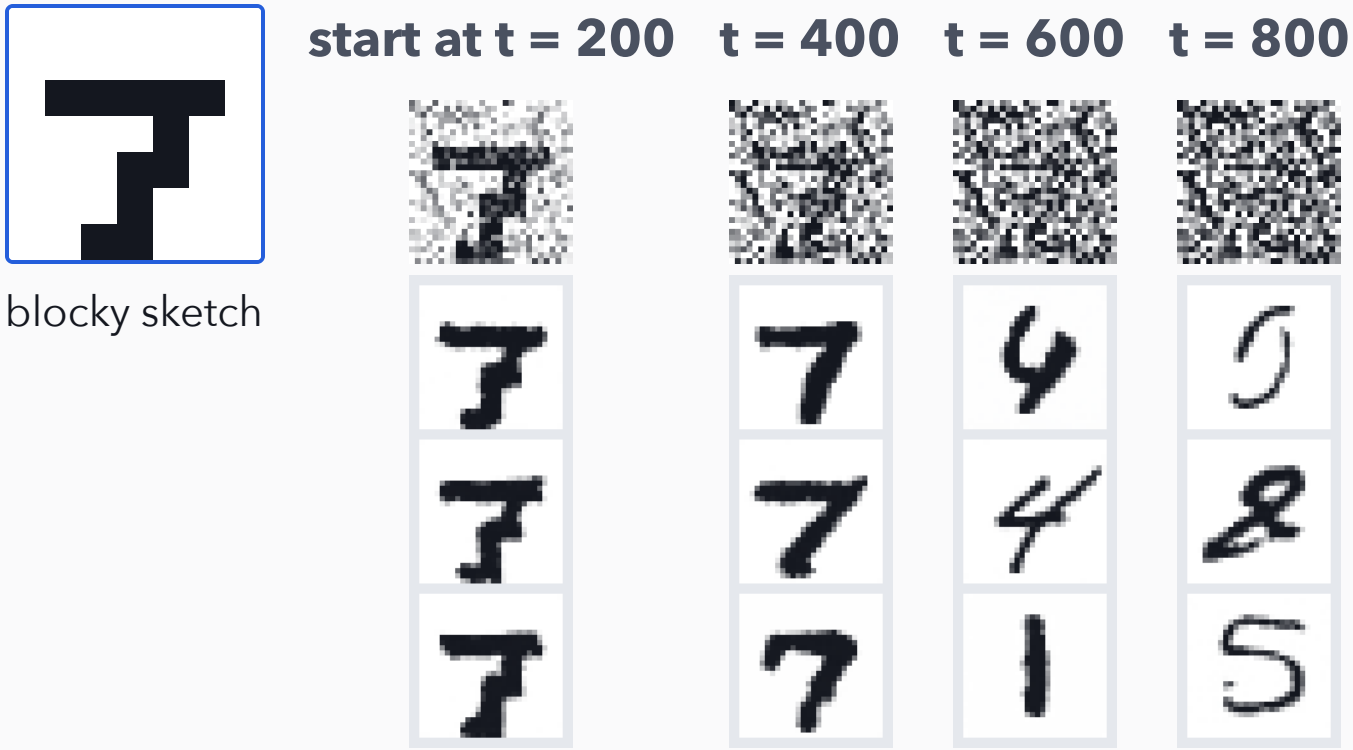
six samples:



No retraining: known pixels come from the forward process, unknown ones from the reverse step. Different seeds give different digits that agree with the top half.

OUR MODEL

Start the reverse chain from a noised sketch



top: the noised sketch $\sqrt{\bar{\alpha}_{t_0}}\, x_{\text{sketch}} + \sqrt{1-\bar{\alpha}_{t_0}}\, \epsilon$ · below: three outputs each

> **Little noise keeps the sketch's blocks; a lot of noise gives a realistic digit that may drift from it. The starting t sets the trade-off.**

THE SAME SAMPLER LOOP FROM PART 3, WITH CONDITIONS ADDED

Guidance and inpainting are a few changed lines

```
x = torch.randn(B, 1, 28, 28)
for t in range(T, 0, -1):
    tt = torch.full((B,), t)
    eps_u = model(x, tt, none)                  # label hidden
    eps_c = model(x, tt, label)                 # label 7
    eps_hat = eps_u + s * (eps_c - eps_u)       # guidance
    c = beta[t] / (1 - alpha_bar[t]).sqrt()
    mean = (x - c * eps_hat) / alpha[t].sqrt()
    x = mean + post_var[t].sqrt() * torch.randn_like(x) if t > 1 else mean
    known_t, _ = corrupt(x_known, torch.full((B,), t - 1))
    x = mask * known_t + (1 - mask) * x        # inpainting
```

guidance: two calls, one line of arithmetic

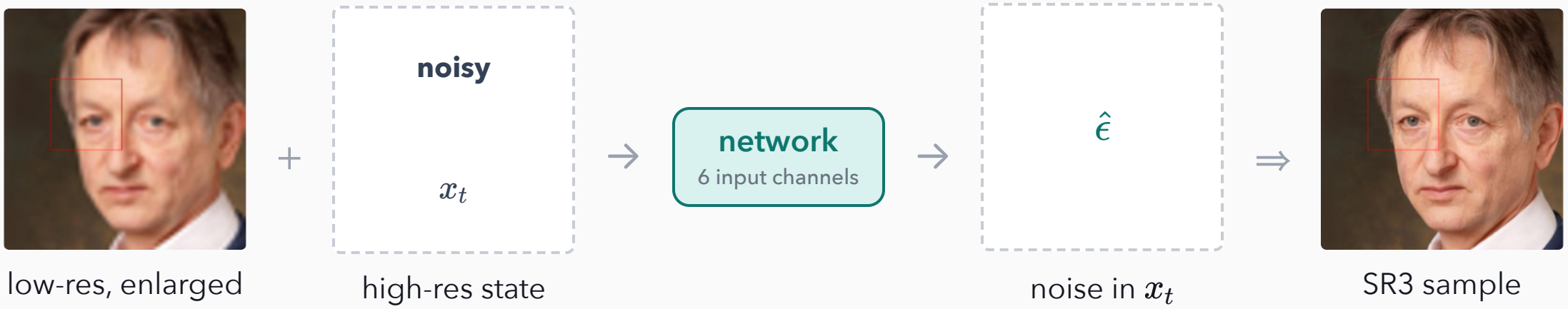
inpainting: overwrite the known pixels, noised to the next step

SDEdit: change only the first line, start from the noised sketch at t_0

The trained network never changes; only how we call it and what we do with its output.

PUBLISHED

Super-resolution: feed the low-res image in as extra channels



Unlike inpainting, this needs a model trained with the low-res input. And, as with our lab images, the added detail is a sample, not a measurement.

SR3, Saharia et al. 2022. Diagram: the conditioning scheme described in the paper (concatenation of the upsampled input).

PART 7 OF 7

Can text be generated by denoising?

Pixels are numbers, so Gaussian noise makes sense; words are not. **Goal of this part:** a different kind of corruption for tokens, with the rest of the recipe unchanged.

- ✓ Corrupt a digit
- ✓ Train: predict the noise
- ✓ Sample: run it backwards
- ✓ Why this loss: the ELBO
- ✓ Inside the network
- ✓ Steer with a condition
- 7 **Text by denoising**

ILLUSTRATIVE

Token IDs are arbitrary labels; arithmetic on them means nothing

sentence	The	dog	chased	the	ball	
vocabulary A	8	24	3	9	11	
vocabulary B	501	2	83	17	4	same sentence, IDs reshuffled

Add the same noise, +1.6, to "dog" and round: in A, 24 → 25.6 → **26**, some unrelated word; in B, 2 → 3.6 → **4 = "ball"**.

corrupt by masking	The	?	chased	?	ball

Numbers near 24 are not words near "dog". For discrete data, corrupt by hiding tokens instead.

ILLUSTRATIVE probabilities

Hide a random fraction t of tokens; predict the hidden ones

clean x_0: The | dog | chased | the | ball

t = 0.4: hide each with prob. 0.4: The | dog | ? | the | ?

slot 3: model's probabilities

chased	0.60
bit	0.20
saw	0.15

slot 5: model's probabilities

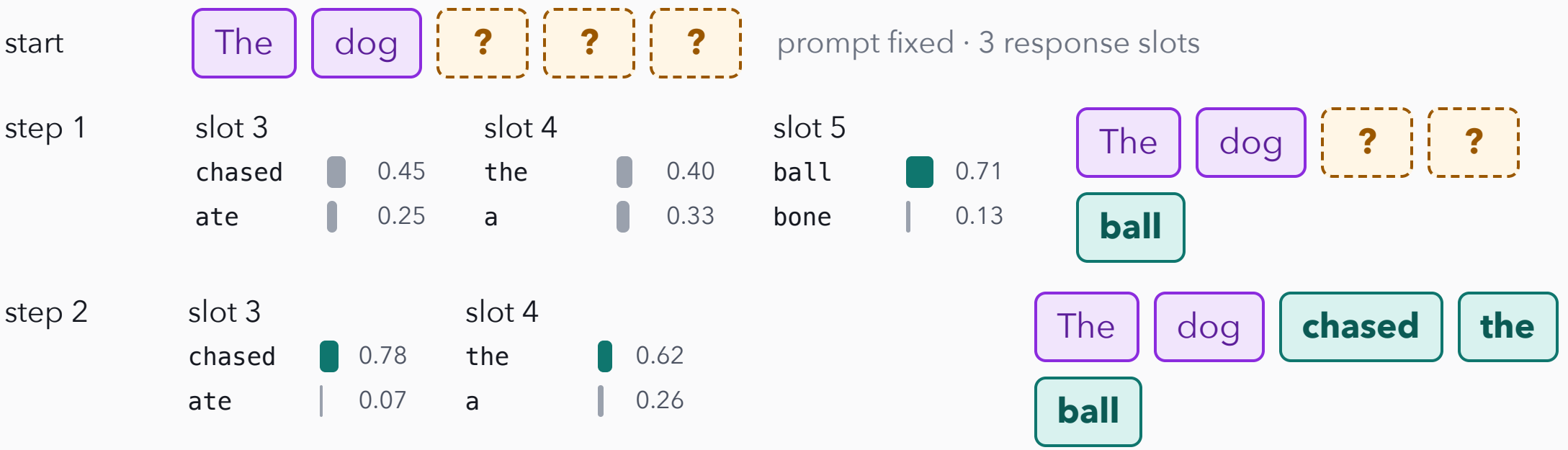
ball	0.70
cat	0.20
car	0.06

loss $= \frac{1}{t}\big(-\log 0.60 - \log 0.70\big) = \frac{1}{0.4}(0.511 + 0.357) = \mathbf{2.17}$ (visible tokens contribute nothing)

Same pattern as images: corrupt by a random amount, predict what was removed. The model sees the whole sequence in both directions.

ILLUSTRATIVE a scripted trace

Start masked; fix the most confident slots first, in any order



> **“ball” was chosen first, then two words at once. Every step predicts all hidden slots, and seeing “ball” made the others easier.**

SAME PROMPT, SAME THREE SLOTS

ILLUSTRATIVE

Two ways to write the same sentence

	AUTOREGRESSIVE (GPT)	MASKED DIFFUSION (LLADA)
one pass predicts	the next token only	every masked token
context used	tokens to the left (causal mask)	all visible tokens, both sides
order	left to right, fixed	any order; set by confidence
passes for our 3 slots	3	here 2; in general a chosen number of steps
length	stop when an end token appears	response length chosen up front

Fewer passes is not automatically faster: each masked-diffusion pass processes the whole sequence, and autoregressive models reuse cached work.

BACK TO THE OPENING SLIDE

Now we can say what each model is trained and run to do

	NOISY OBJECT	CONDITION C	WHAT CHANGES IN OUR RECIPE
 text → image	image latent	77 text vectors	cross-attention to the prompt; classifier-free guidance
 fill a region	pixels	the known pixels	paste noised known pixels back at every step (no retraining)
 painting → photo	pixels	the painting	start the reverse chain at t_0 from the noised painting
 super-resolution	high-res pixels	low-res image	extra input channels; trained on image pairs
 edges → image	image latent	edge map + prompt	a trainable copy of the encoder reads the edges (ControlNet)
 one particular dog	image latent	prompt with a rare word	fine-tune the denoiser on 3–5 photos first (DreamBooth)
The dog ? ? text	tokens	the prompt	mask instead of Gaussian noise; cross-entropy instead of MSE

> **One idea throughout: corrupt, learn to undo one step, then repeat from noise, with the condition as an extra input.**

The whole method on one slide

1. Training input: $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$ for a random t — our pixel: 0.8·1 + 0.6·(−0.5) = 0.5
2. Target: the noise ϵ, with squared error — best possible guess: an average, −0.24
3. Sampling: from $x_T \sim \mathcal{N}(0, I)$, repeat mean + $\sigma_t z$ down to t = 1 — one step: 0.5 → 0.708 → 0.808
4. Why: each ELBO term is a weighted noise MSE; DDPM drops the weights — our pixel: $0.625 \times 0.1^2 = 0.00625$

> **Removing noise creates an image because each small step is a sample from the learned reverse distribution, and the steps add up.**

CHECK YOURSELF

Three questions to answer before next class

1. Does training require running the reverse chain?

   No. One random t, one jump-formula input, one network call, one MSE per example.

2. Why is the guess $\hat{x}_0$ different from the next state x_{t-1}?

   $\hat{x}_0$ is a full jump to a clean estimate, unreliable at high noise; x_{t-1} moves only a little towards it, adds fresh noise, and is the state fed back in.

3. What changes when the data are tokens?

   Corruption becomes masking and the loss becomes cross-entropy on masked positions; the sampler unmasks confident slots over several passes.

APPENDIX

Derivations and extensions

A1 adding Gaussians · A2 the training posterior · A3 the bound · A4 ϵ, x_0 and v · A5 schedules · A6 DDIM · A7 the score view · A8 flow matching · A9 masked-diffusion objective · A10 attention heads · A11 reporting a generated example

COMPUTED

Independent Gaussians add: the step from t − 1 to t

assume $x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\, x_0 + \sqrt{1-\bar{\alpha}_{t-1}}\, \epsilon'$

one step $x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{\beta_t}\, \eta_t = \sqrt{\alpha_t \bar{\alpha}_{t-1}}\, x_0 + \sqrt{\alpha_t(1-\bar{\alpha}_{t-1})}\, \epsilon' + \sqrt{\beta_t}\, \eta_t$

variances $\alpha_t(1-\bar{\alpha}_{t-1}) + (1-\alpha_t) = 1 - \alpha_t\bar{\alpha}_{t-1} = 1 - \bar{\alpha}_t$

so $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon$, true for t = 1, hence for all t

Toy check at t = 2: 0.8 × 0.2 + 0.2 = 0.36 = 1 − 0.64 (Part 1).

COMPUTED

Knowing x_0 makes the reverse step a known Gaussian

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\tilde{\mu}_t,\ \tilde{\beta}_t I), \qquad \tilde{\beta}_t = \beta_t \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$$

$$\tilde{\mu}_t = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}\, x_0 + \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\, x_t$$

0.497 × 1 (toy: true x_0) + 0.497 × 0.5 (x_2) = **0.745** ($\tilde{\mu}$)

0.497 × 0.925 (the guess $\hat{x}_0$ instead) + 0.497 × 0.5 = **0.708** (μ_θ)

> The model's reverse mean is the true posterior mean with $\hat{x}_0$ plugged in; rewriting $\hat{x}_0$ in terms of $\hat{\epsilon}$ gives the formula of Part 3.

Bayes on each forward step makes the bound telescope

$$-\log p_\theta(x_0) \;\le\; \mathbb{E}_q\Big[-\log p(x_T) - \sum_{t\ge 1}\log \frac{p_\theta(x_{t-1}\mid x_t)}{q(x_t\mid x_{t-1})}\Big]$$

Jensen, as in Lecture I, with $p_\theta(x_{0:T}) = p(x_T)\prod_t p_\theta(x_{t-1}\mid x_t)$ and $q(x_{1:T}\mid x_0) = \prod_t q(x_t\mid x_{t-1})$

for $t \ge 2$: $\quad q(x_t\mid x_{t-1}) = \dfrac{q(x_{t-1}\mid x_t, x_0)\; q(x_t\mid x_0)}{q(x_{t-1}\mid x_0)}$ (Bayes, using the Markov property)

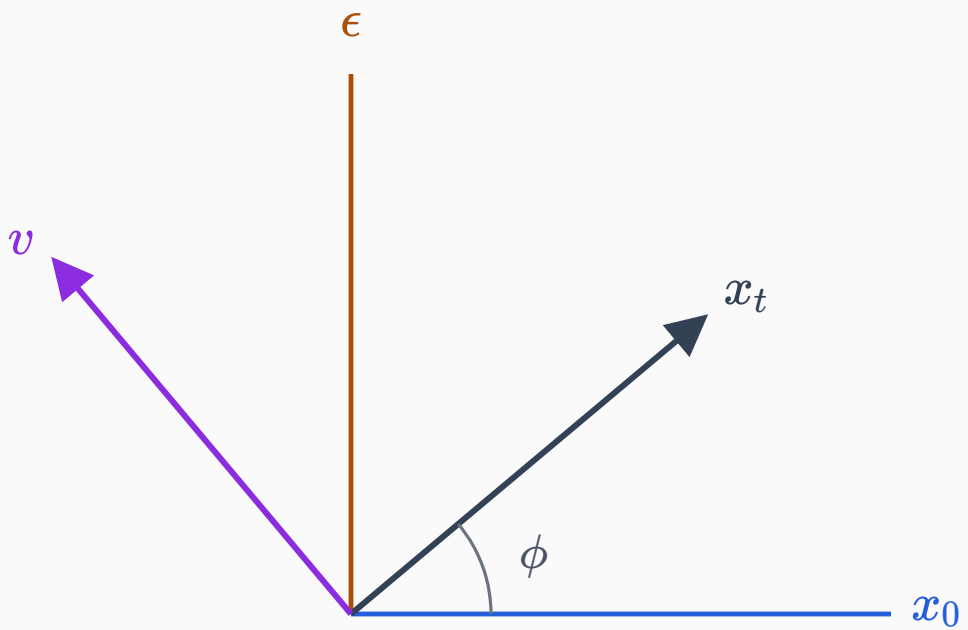
$$\text{so} \;=\; \underbrace{D_{KL}(q(x_T\mid x_0)\,\|\,p(x_T))}_{L_T} \;-\; \underbrace{\mathbb{E}_q \log p_\theta(x_0\mid x_1)}_{L_0}$$

$$+\; \sum_{t=2}^{T} \underbrace{\mathbb{E}_q D_{KL}(q(x_{t-1}\mid x_t, x_0)\,\|\,p_\theta(x_{t-1}\mid x_t))}_{L_{t-1}}$$

> **The ratios $q(x_t\mid x_0)/q(x_{t-1}\mid x_0)$ cancel in a chain, leaving only $q(x_T\mid x_0)$: this is the decomposition used in Part 2.**

COMPUTED

x_t and v are x_0 and ϵ rotated by an angle that grows with t



$$\cos\phi = \sqrt{\bar{\alpha}_t}, \quad \sin\phi = \sqrt{1-\bar{\alpha}_t}$$

$$x_t = \cos\phi\, x_0 + \sin\phi\, \epsilon$$

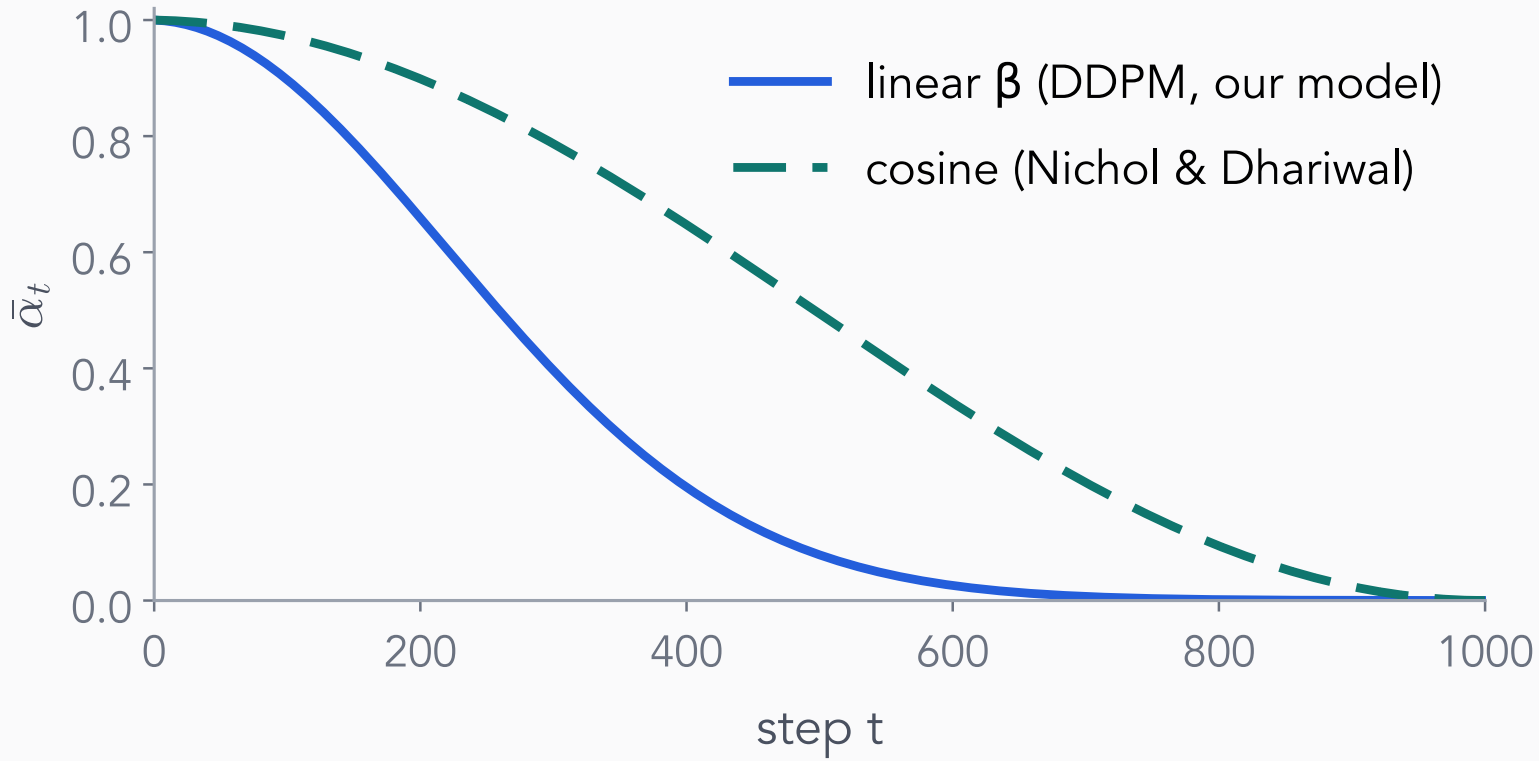
$$v = \cos\phi\, \epsilon - \sin\phi\, x_0$$

$$x_0 = \cos\phi\, x_t - \sin\phi\, v, \qquad \epsilon = \sin\phi\, x_t + \cos\phi\, v$$

Given x_t, predicting any one of ϵ, x_0, v determines the others; the sampler must use the matching conversion.

COMPUTED

The cosine schedule spends fewer steps on near-pure noise



> **With the linear schedule, $\bar{\alpha}$ is nearly 0 for the last ~40% of steps (Part 1's identical panels). The cosine schedule removes signal more evenly.**

Cosine schedule: Nichol & Dhariwal 2021, $\bar{\alpha}_t = f(t)/f(0)$, $f(t) = \cos^2\left(\frac{t/T+0.008}{1.008} \cdot \frac{\pi}{2}\right)$.

OUR MODEL

DDIM: jump between chosen steps using the clean guess

$$x_s = \sqrt{\bar{\alpha}_s}\,\hat{x}_0 + \sqrt{1-\bar{\alpha}_s}\,\hat{\epsilon}, \qquad s < t$$

no fresh noise: re-noise the clean guess with the predicted noise

DDPM, 1000 calls



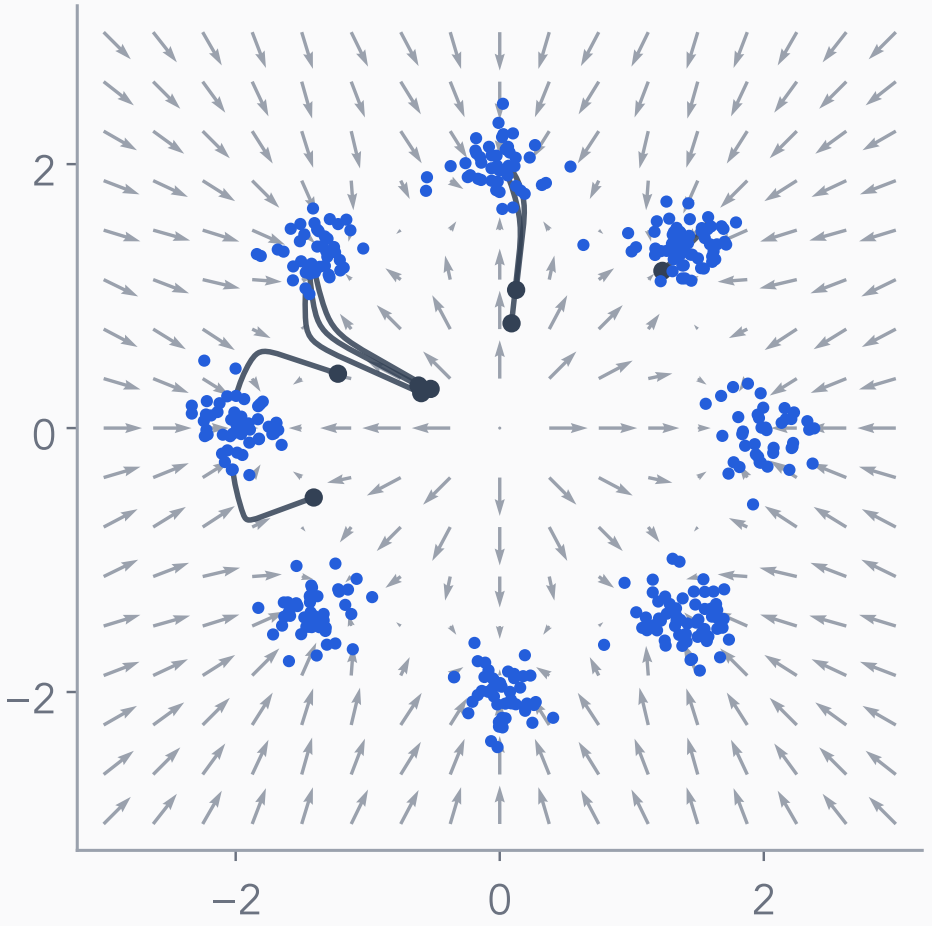
DDIM, 50 calls



> **Same weights, same starting noise, 20× fewer calls. Deleting steps from DDPM's loop would not work: DDIM defines a different transition.**

COMPUTED exact score, no network

Predicting noise is learning which way density increases



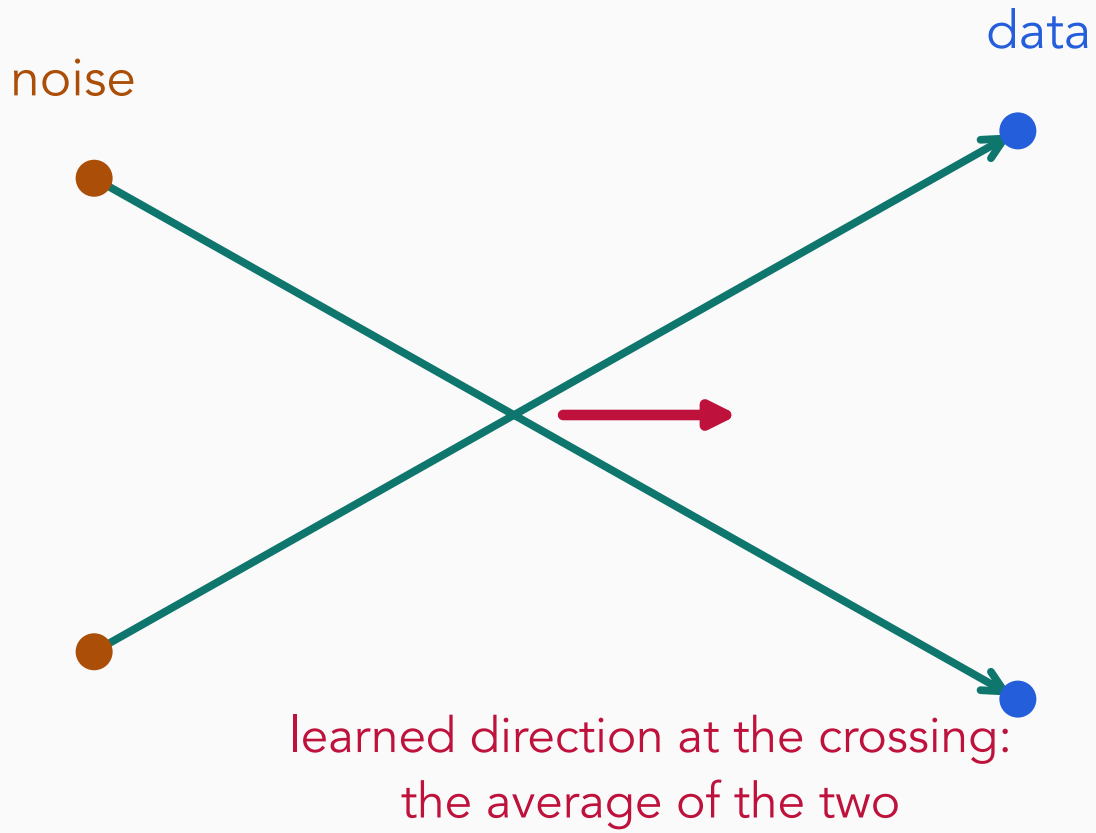
$$\nabla_x \log q_t(x) = -\frac{\mathbb{E}[\epsilon \mid x_t = x]}{\sqrt{1-\bar{\alpha}_t}}$$

For the cluster data this score is known exactly, so we can run the sampler with the **exact** ϵ instead of a network.

Arrows: the score at t = 150. Lines: 8 reverse trajectories (deterministic DDIM steps) from noise; dots: 400 final samples, which land on the clusters.

COMPUTED

Learn a velocity along straight lines from noise to data



$$x_\tau = (1-\tau)\, x_{\text{noise}} + \tau\, x_{\text{data}}$$

$$u = x_{\text{data}} - x_{\text{noise}}$$

$\min_\theta \ \mathbb{E}\, \|v_\theta(x_\tau, \tau) - u\|^2$, then integrate $\dfrac{dx}{d\tau} = v_\theta$

Where straight lines cross, the network can only learn their **average** direction, so the learned paths curve.

LLaDA's loss and its low-confidence remasking

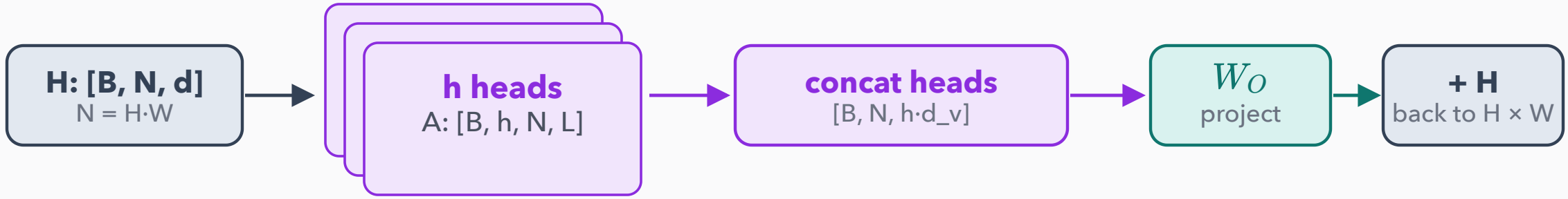
$$\mathcal{L}(\theta) = -\,\mathbb{E}_{t,x_0,x_t}\left[\frac{1}{t}\sum_{i=1}^{L}\mathbf{1}[x_t^i = \texttt{MASK}]\,\log p_\theta(x_0^i \mid x_t)\right]$$

$t \sim U(0,1]$: each token masked independently with probability t

1. Choose a response length and a number of steps; mask every response slot.
2. Each step: predict all masked slots in one pass.
3. Keep the most confident predictions (the schedule says how many); re-mask the rest.
4. Prompt tokens are never masked during fine-tuning or generation.

Nie et al., Large Language Diffusion Models (LLaDA), 2025.

Each head attends separately; then concatenate and project



$$Q : [B, h, N, d_k], \quad K : [B, h, L, d_k], \quad V : [B, h, L, d_v]$$

$$A = \mathrm{softmax}_L\left(QK^\top / \sqrt{d_k}\right) : [B, h, N, L]$$

> Softmax over the L text positions, for every image position in every head; then W_O maps $h\,d_v$ back to the image feature width.

Say where every image came from

BADGE	WHAT IT MEANS	WHAT TO REPORT
PUBLISHED	a figure from a paper or project page	source, figure, what the authors say was supplied (prompt, input)
OUR MODEL	produced by a model trained for this lecture	checkpoint, steps, seed, sampler, whether selected
COMPUTED	exact arithmetic or the fixed forward process	the formula and the inputs
ILLUSTRATIVE	hand-chosen numbers to show a mechanism	that it is not a measured result

Our models: pixel DDPM 814,145 parameters, 30000 steps, batch 128, seed 667, EMA 0.999, linear $\beta \in [10^{-4}, 0.02]$, $T = 1000$, posterior variance; class-conditional copy with label dropout 0.1; $4 \times 7 \times 7$ autoencoder + latent DDPM. Scripts: `scripts/train_models.py`, `scripts/make_figures.py`.