

U1.L1 — Coupling Flows on 2D Toys

Flow-Based Generative Models · UFRJ · 2026.2

01 | Orientation and handover

Where this session sits

Session	What happened
U1.T1	a coupling layer has a cheap inverse and a log-det that is a sum
U1.L1 — today	you write it, check it, train it, look at it, read the library's version
U1.T2	what such a stack can and cannot represent

Today is the first lab of the joint phase, and the first time a generative model is written from nothing and trained.

Three objects you leave with

1. A **bijection** you wrote — `AffineCoupling`, and a Flow that stacks it.
2. **Exact density evaluation** — `Flow.log_prob`, the change-of-variables theorem as code.
3. An **independent Jacobian witness** — `autodiff`, per example, `float64`, before and after training.

And one thing deliberately not on the list: an explanation of the last picture of the day.

Given code, three modules

Module	Built in	What this lab uses
common/train.py	Uo.L1, Uo.L2	set_seed; Trainer and its .fit
common/toy2d.py	today	three frozen generators, two picture helpers
common/coupling.py	today	the reference you rebuild

The Trainer: five-step loop, schedule stepped **per step**, validation under no_grad every epoch, last.pt and best.pt, a CSV log. It expects (x, y) batches; a flow has no labels, so y is a placeholder and the accuracy reads nan — an interface preserved, not a defect.

Where the curves go

MLflow, **deployed locally**: `sqlite:///mlflow.db` next to the notebook, plus `mlartifacts/`. Portable — an offline bundle stands alone. Not a production recommendation.

```
mlflow ui --backend-store-uri sqlite:///mlflow.db
```

One departure, measured: per-**step** logging turned a 14 s training into 48 s (a step is under a millisecond; a SQLite write is a few). So the loop runs without the tracker, and the two NLL curves (negative log-likelihood: the loss, on train and on validation) are logged **per epoch**, after the loop, with the config and the tags.

The sanity ritual, third appearance

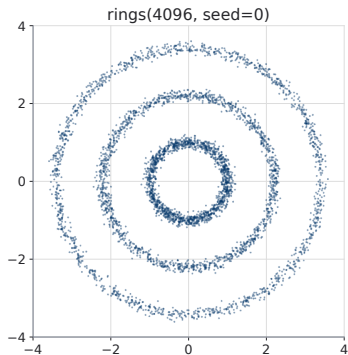
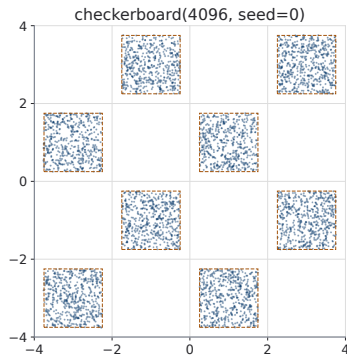
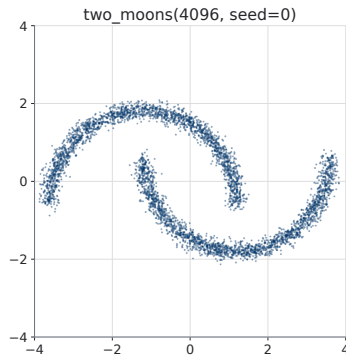
$$\mathbf{v}^\top \nabla_\theta L \approx \frac{L(\theta + \varepsilon \mathbf{v}) - L(\theta - \varepsilon \mathbf{v})}{2\varepsilon}, \quad \|\mathbf{v}\| = 1$$

Float64, one fixed unit direction, mixed tolerance $\text{atol} + \text{rtol} \cdot \max(|d_{\text{AD}}|, |d_{\text{FD}}|)$.
Two extra forward evaluations, whatever the parameter count.

Today's sibling: the suspect is a *log-determinant*. Analytic sum of scales, against a Jacobian that autodiff assembles independently, per example.

02 | toy2d: the datasets, frozen today

Three generators, one contract



The contract

ARTIFACT TOY2D-DATASETS – FROZEN

```
two_moons / checkerboard / rings (n, seed)
  -> Tensor (n, 2), float32; EXTENT = (-4.0, 4.0)
density_heatmap(log_prob_fn, extent, n, ax)
  -> (xs, ys, logp)
warped_grid(generative_fn, extent, ax) -> lines
```

1. **Pure seeded functions.** Same seed, same pinned environment, same tensor, bit for bit.
2. **Float32**, shape (n, 2).
3. **Literal zero-density gaps** in the checkerboard: eight separated squares of side 1.5, no jitter.

Downstream sessions import this module by name and never redefine a generator.

The gapped checkerboard is a witness

- Mixture of **eight** squares of side 1.5, centred in the black cells of a 4×4 board on $[-4, 4]^2$.
- Nearest corners of two same-colour squares: $0.5\sqrt{2} \approx 0.71$ apart. White cells: no mass at all.
- The support of q is a union of eight **disjoint** closed squares; the density is exactly $1/18$ on it.

That is a mathematical statement about literal support, and it is there because U1.T2 needs one. **Today you only observe** what a fitted density does in the gaps.

03 | An affine coupling layer, from scratch

The layer, generative direction

$$\text{forward: } y_A = x_A, \quad y_B = x_B \odot e^{s_\theta(x_A)} + t_\theta(x_A), \\ \log |\det J_\psi| = \sum_j s_\theta(x_A)_j$$

$$\text{inverse: } x_A = y_A, \quad x_B = (y_B - t_\theta(y_A)) \odot e^{-s_\theta(y_A)}, \\ \log |\det J_{\psi^{-1}}| = -\sum_j s_\theta(y_A)_j$$

Both sums come off the **same** conditioner pass: $x_A = y_A$ is on both sides. In code, z is the source-side input (z_{k-1} of the T1 chain; $z_0 = x_0$), x the data-side output, `mask` marks x_A with `True`.

The interface is frozen

ARTIFACT COUPLING-LAYER-SCRATCH

```
AffineCoupling(d, mask, hidden=(64, 64), s_cap=2.0)
    forward(z) -> (x, logdet_fwd) # psi: source -> data
    inverse(x) -> (z, logdet_inv) # psi^-1: data -> src
    # logdet_inv == -logdet_fwd, PER EXAMPLE
Flow(layers): forward / inverse / log_prob(x) / sample(n)
Permute(perm): both log-dets exactly 0
```

TODO 2.1 writes `_heads`, `forward`, `inverse`; the constructor and the conditioner (an MLP $\mathbb{R}^{d_A} \rightarrow \mathbb{R}^{2d_B}$, two heads) are given. Checkpoint: `x[:, mask] == z[:, mask]`, and the log-det at init is **not** zero.

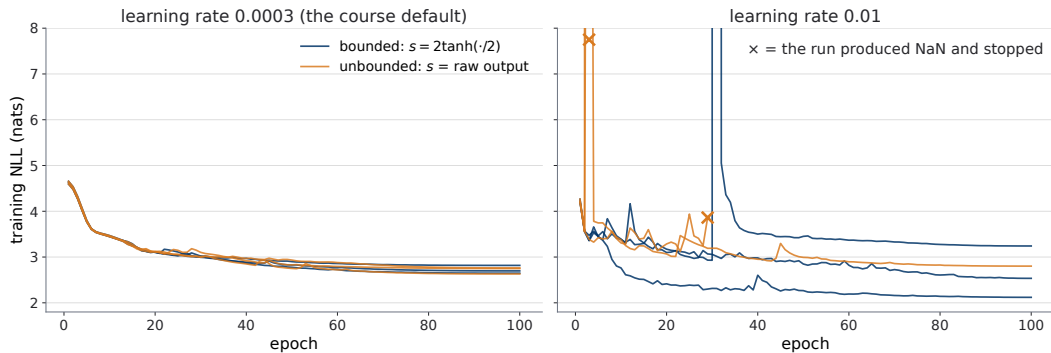
Bound the scale — the policy

$$s = s_{\max} \tanh(\text{raw}/s_{\max}), \quad s_{\max} = 2$$

So $|s| < 2$: every layer's volume factor e^s stays in $[0.14, 7.4]$. Near zero the map is the identity; far from zero it saturates, so a random conditioner can neither overflow e^s nor divide the inverse by ≈ 0 .

U1.T1 promised to show what the loss looks like when you forget. Next slide, measured.

Bound the scale — the measurement



Why the head is not zero

- The recipe card zero-opts a last layer where “output ≈ 0 at start” is sensible — and an identity-at-start coupling layer **is** sensible.
- But a zero head gives $s \equiv 0$: analytic log-det 0, autodiff log-det 0, perfect agreement **on nothing**. Uo.L3 and Uo.L4 both met this.
- So the head starts **small and nonzero** (default init $\times 0.1$). Measured at init, six layers, 64 points: $\max |\log |\det J|| = 0.020$.

A check that cannot fail has verified nothing. The notebook asserts the log-det is nonzero **before** trusting any agreement.

TODO 2.2 — the independent witness

PER-EXAMPLE JACOBIAN, FLOAT64

```
J = torch.func.vmap(torch.func.jacrev(single))(z) # B,d,d
sign, logabsdet = torch.linalg.slogdet(J)         # (B,)
```

`single` sends one point $(d,)$ to one point $(d,)$. Accept under $\text{atol} = 1\text{e-}10$, $\text{rtol} = 1\text{e-}8$ on the largest log-det; assert $\text{sign} > 0$.

Measured, 64 points: gap 5.3×10^{-16} (tol 3.0×10^{-10}) untrained; 5.1×10^{-15} (tol 2.1×10^{-8}) after training, where $\max |\log |\det J|| = 2.11$.

The pitfall: one determinant of the batch

`torch.autograd.functional.jacobian(f, z)` on the **whole batch** returns the Jacobian of the batched map $\mathbb{R}^{Bd} \rightarrow \mathbb{R}^{Bd}$, shape (B, d, B, d) .

Measured, $B = 64$:

Quantity	Value
largest off-diagonal block entry	0.0
slogdet of the batched map	0.24336826533634
sum of the 64 per-example log-dets	0.24336826533634

One number for the whole batch. It cannot say **which** example is wrong; its plain det is a product of B factors and overflows for a trained flow; it costs B times the memory.

TODO 2.3 — both round trips

Per example, float64:

- $\text{inverse}(\text{forward}(z)) \approx z$
- $\text{forward}(\text{inverse}(x)) \approx x$
- $\text{logdet_fwd} + \text{logdet_inv} \approx 0$ at matching points

Measured, untrained: 8.9×10^{-16} , 8.9×10^{-16} , 2.8×10^{-17} . Trained: 9.1×10^{-15} , 2.7×10^{-15} , 1.0×10^{-15} .

The third line is U1.T1's Exercise 3, executed: the inverse pass reads the same sum with the opposite sign.

04 | Stack, and train

TODO 3.1 — compose, and the sign

Flow.forward: layers in order, log-dets **add**. Flow.inverse: layers **reversed**, each through its own inverse, log-dets add. Masks alternate.

$$\log p_{\theta}(x) = \log p_o(z) + \log |\det J_{\psi^{-1}}(x)|, \quad z = \psi^{-1}(x)$$

In code: `z, logdet_inv = inverse(x); return log_po(z) + logdet_inv`. That is U1.T1's $\log p_o(z_o) - \sum_k \log |\det J_{\psi_k}(z_{k-1})|$ with the minus sign absorbed into `logdet_inv`. Written next to the implementation, so the sign is never implicit.

A permutation is free — in code

U1.T1's Exercise 1 asked you to check in your head that a coordinate swap contributes 0 and undoes itself. Checkpoint 3.1 checks it in code:

```
mixed = Flow([layer_1, Permute([1, 0]), layer_2], 2)
y, ld = mixed(z)          # the swap adds exactly 0.0
z_back, _ = mixed.inverse(y)
assert (z_back - z).abs().max() < 1e-12
```

Alternating masks and inserted permutations are two spellings of the same idea: after two layers every coordinate has moved.

TODO 3.2 — the closure is all you write

$$\mathcal{L}(\theta) = -\mathbb{E}_{x_1 \sim q} [\log p_{\theta}(x_1)] \approx -\frac{1}{B} \sum_{i=1}^B \log p_{\theta}(x_1^{(i)})$$

```
class LogProbModel(nn.Module):    # Trainer calls model(x)
    def forward(self, x):
        return self.flow.log_prob(x)

class NLL(nn.Module):             # loss_fn(model(x), y)
    def forward(self, logp, y):
        return -logp.mean()
```

Everything else is the given Trainer.

Settings and budget

- **Optimizer, schedule** — the course defaults: AdamW at 3×10^{-4} , weight decay 0.01; warmup over 5% of the steps, then cosine, **per step**.
- **Model** — $K = 6$ affine coupling layers, conditioners of width 64, $s_{\max} = 2$: 26,508 parameters.
- **Budget** — 500 epochs $\times$ 16 steps = 8000 steps at batch 256. Measured: 69 s on a laptop core.
- **Data** — train `two_moons(4096, 0)`, validation `two_moons(1024, 1)`. Selection on validation; there is no test set.

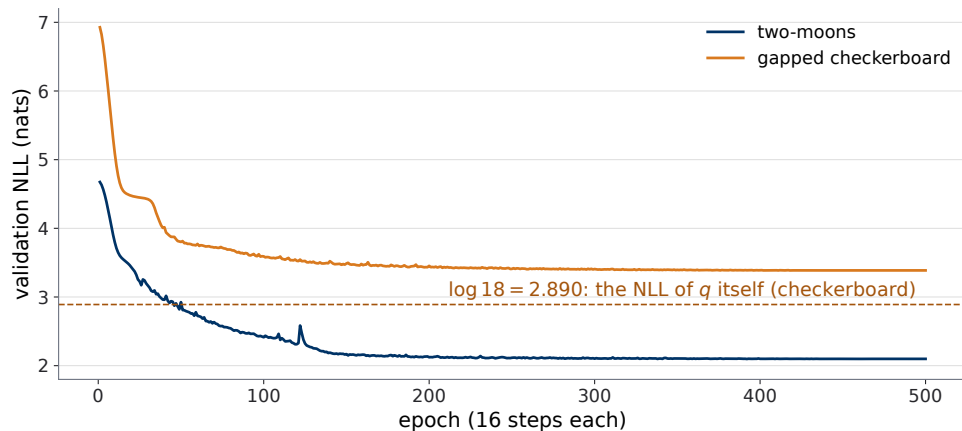
The relative checkpoint

Pass if: the validation NLL at the end beats the first epoch's by **more than one nat**, and the trained stack still passes the witness and the round trips.

Run	Time	Val NLL, epoch 1	Epoch 500
two-moons	69 s	4.674	2.099
gapped checkerboard	69 s	6.927	3.387

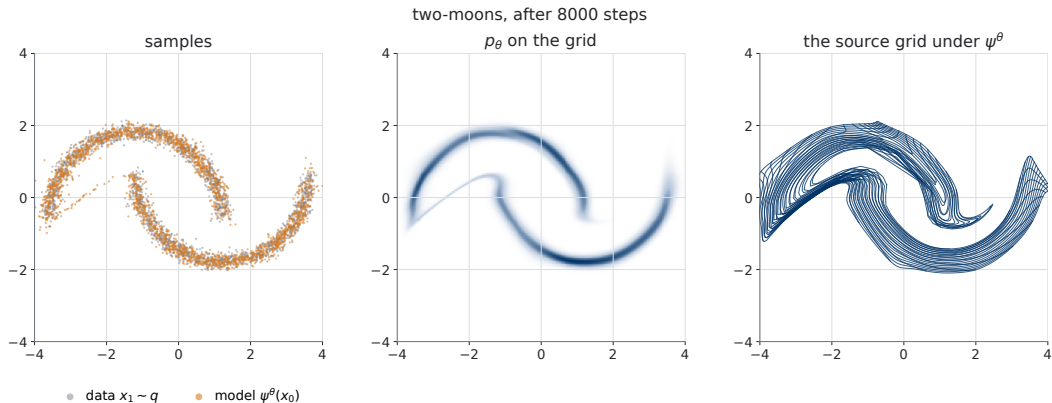
No absolute threshold: an absolute number is a promise about your hardware that nobody validated. **TODO 3.3:** same code, same budget, the checkerboard — one line changes.

Checkpoint — the curves



05 | Look at what you built

Two-moons: three views



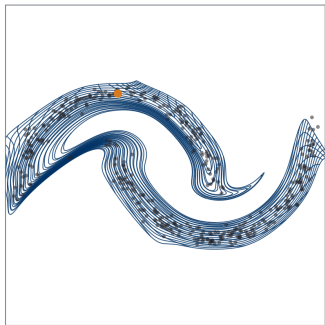
The warped grid is T1's picture

- U1.T1 pushed **one small cube** through a diffeomorphism and read the density change off the volume change.
- This panel pushes the **whole source grid** through the learned diffeomorphism; the same reading holds cell by cell: large cell, density divided; crushed cell, density multiplied.
- The grid is one connected sheet, folded and stretched, **never torn**. A diffeomorphism cannot tear.

Keep that last sentence for the next figure.

The source grid, folded into the moons

The source grid, folded into the moons — one layer at a time



six layers, one connected sheet — stretched and folded, never torn

after all 6 layers

$\log|\det J|$ so far, at the orange sample

-1.105

autodiff $\log|\det J|$ at the same sample

-1.105

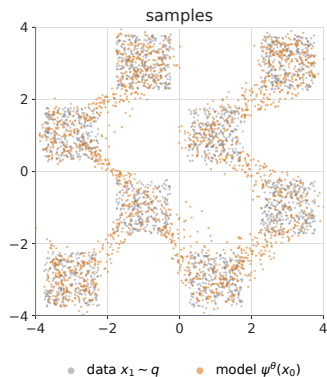
telescoped sum and Jacobian agree to $8e-16$

$\log p_\theta(x) = \log p_0(z_0) - \log |\det J_\psi(z_0)|$

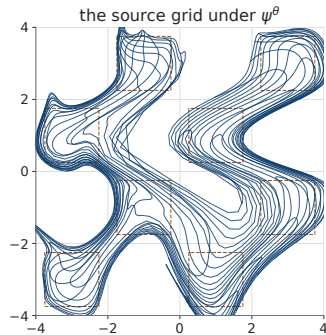
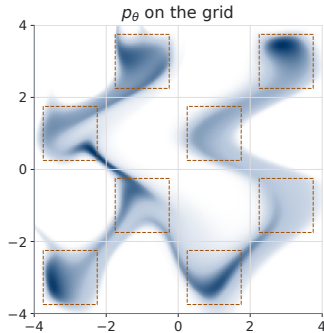
$-1.737 = -2.842 - (-1.105)$

Animated in the web deck — final frame shown.

The gapped checkerboard: three views



gapped checkerboard, after 8000 steps



The question, and only the question

! Important

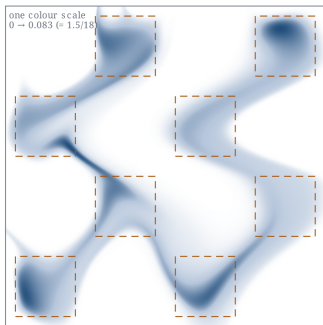
What density does the model place in the hard-zero gaps, and what happens to the grid near those regions?

Measured on the reference model, 400×400 grid:

- 29.8% of the model's mass lies **outside** the eight squares, where q is exactly zero;
- along the corner-to-corner segment between two diagonal neighbours the density never drops below 0.040 (uniform level inside a square: $1/18 \approx 0.056$).

The bridges appear, and stay

The bridges appear, and stay — the checkerboard during training



What density does the model place in the hard-zero gaps,
and what happens to the grid near those regions? — explained in U1.T2

epoch 500

of 500, 16 steps each

validation NLL

3.387

$\log 18 = 2.890$: the NLL of q itself

mass outside the eight squares

29.8 %



Animated in the web deck — final frame shown.

The observation, and where it goes

- The grid is stretched **across** every gap — one connected sheet, eight bulges, material in between.
- The validation NLL stops 0.50 nats above $\log 18$, the NLL of q itself.

Explained in U1.T2. The solution records the observation and says exactly that. Do not read the size of the bridge as a theorem: what is exactly impossible for this model, and what is an effect of finite capacity or of optimization, are separated in that session.

06 | The library cross-check

Board 1 — two costs, two dialects

course $\psi: z \rightarrow x$ · **library** $f = \psi^{-1}: x \rightarrow z$

Operation	Direction of ψ^θ	Needs, per layer
Sampling	forward, $x_0 \rightarrow x_1$	ψ_k
Density / training	inverse, $x_1 \rightarrow x_0$	ψ_k^{-1} and $\log \det J_{\psi_k} $

Zuko exposes the **density** transform as its primary object. Never flip the course convention to match a library; write $f = \psi^{-1}$ and translate.

Same Trainer, one token changes

Zuko flows are `nn.Modules` whose **call** returns a distribution: `flow()` is p_θ .

```
zflow = zuko.flows.RealNVP(features=2, transforms=6,  
                           hidden_features=(64, 64))
```

```
class ZukoLogProb(nn.Module):  
    def forward(self, x):  
        return self.flow().log_prob(x)  # flow(), then
```

Same data, same seeds, same optimizer, same schedule, same Trainer, same budget. **That is the point of a cross-check.**

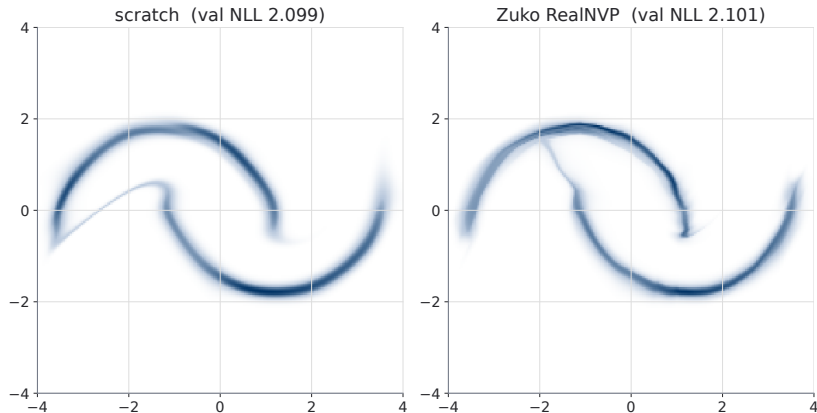
The direction, asserted

LIBRARY DIALECT, TRANSLATED

	Map	Direction	Used by
course	ψ	$z \rightarrow x$	sample, your forward
<code>flow.transform()</code>	$f = \psi^{-1}$	$x \rightarrow z$	log_prob
<code>flow.transform().inv</code>	ψ	$z \rightarrow x$	sample

`f.log_abs_det_jacobian(x, z)` is $\log |\det J_{\psi^{-1}}(x)|$ — your `logdet_inv`.
Checked, not trusted: `log_p0(f(x)) + ladj` against `flow().log_prob(x)`
on 1024 points, gap 9.5×10^{-7} (float32); `f.inv(f(x))` returns `x` to 3.3×10^{-6} .

The cross-check, passed



TODO 5.2 — source reading: two classes, one layer

Zuko splits the layer in two:

- the **lazy** class, `GeneralCouplingTransform` — owns the mask and the conditioner (the *hypernetwork*), builds a transform on demand;
- the **transform**, `CouplingTransform` — does the arithmetic, and delegates the per-coordinate map to the **affine map**, `MonotonicAffineTransform`.

Your one class maps onto the three of them. The next slide says what to find in each.

TODO 5.2 — four things to find

Your code	In Zuko	Find
mask	lazy: <code>__init__</code> ; transform: <code>split, merge</code>	which value is the untouched half; how NICE alternates it
<code>_heads</code> <code>inverse</code>	lazy: <code>meta, hyper</code> transform and affine map: <code>_inverse</code>	what <code>meta</code> returns which of <code>_call</code> / <code>_inverse</code> is ψ
the two log-dets	<code>log_abs_det_jacobian</code> ; <code>call_and_ladj</code>	where the sum is taken; how the log-scale is bounded

Line anchors live in the notebook, generated from the pinned zuko 1.5.0 at build time. TODO 5.3 is reserved (only if U1.L2 is cut).

07 | JAX, and the close

The same map in JAX

```
def psi(z):                                # ONE example
    x = z
    for first_is_a, ws, s_cap in params:
        ia, ib = (0, 1) if first_is_a else (1, 0)
        raw = mlp(ws, x[ia:ia + 1])
        s = s_cap * jnp.tanh(raw[0] / s_cap)
        x = x.at[ib].set(x[ib] * jnp.exp(s) + raw[1])
    return x

J = jax.vmap(jax.jacfwd(psi))(z)           # (B, 2, 2)
_, logabsdet = jnp.linalg.slogdet(J)
```

Trained torch weights, copied over. Agreement with torch on 64 points: log-dets to 1.1×10^{-13} , forward maps to 3.9×10^{-15} .

Three objects you own

Object	Checked by
a bijection : <code>AffineCoupling + Flow</code>	both round trips, per example, before and after training
exact density evaluation : <code>Flow.log_prob</code>	the sign check; the training curves; a library within 0.002 nats
an independent Jacobian witness	<code>torch.func</code> and JAX, per example, float64

And one thing you do not own yet: an explanation of the checkerboard's gaps.
29.8% of the mass outside the support; 0.50 nats above log 18. **U1.T2 opens there.**

What comes next

- **Next session (U1.T2):** it opens with today's checkerboard and asks what a flow built from diffeomorphisms can and cannot represent exactly; then the rest of the zoo, and the depth question U1.T1 left open.
- **Before then:** finish TODO 4.3 in words, and use the notebook's last cell to check your PS1.4(b) — the reference agrees with `Flow.log_prob` to 1.8×10^{-15} .

Sources for today: (Dinh, Sohl-Dickstein, and Bengio 2017, secs. 3.2–3.3, §3.5), (Papamakarios et al. 2021, sec. 3.1.2), (Rozet et al. 2022).

References

- Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio. 2017. "Density Estimation Using Real NVP." In *International Conference on Learning Representations*.
<https://arxiv.org/abs/1605.08803>.
- Papamakarios, George, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. 2021. "Normalizing Flows for Probabilistic Modeling and Inference." *Journal of Machine Learning Research* 22 (57): 1–64.
<https://arxiv.org/abs/1912.02762>.
- Rozet, François et al. 2022. "Zuko: Normalizing Flows in PyTorch."
<https://doi.org/10.5281/zenodo.7625672>.