Experiment 2.9.3: when anchoring fails

> Every failing run anchors first and then comes apart during the high learning-rate plateau. Halving the peak learning rate removes the failures.

Ex-2.9.2 identified two components to the variance in our ablation scores. The redistribution was fixed: fallback control gives the intervention a response we design ahead of time. The other part stayed open: On some seeds the concept never landed cleanly on its axis.

Our guess was that the regularizer schedule is incompatible with some initializations: some starting weights simply cannot be anchored by this schedule. If that were true, the remedy would be a schedule search per seed or per init, which is expensive and gets worse as models grow.

This experiment tests that, and it doesn't hold up. There are three arms, all built on the small color autoencoder from ex-2.9.1.

- Trajectories: retrain the base arm of ex-2.9.2 (the same 32 seeds), recording anchor progress, leakage, and reconstruction error at every step. When do failures happen, relative to the schedule?
- Attribution: separate the two sources of randomness, the model init and the batch/label stream (16 inits × 8 streams, including the two inits whose earlier runs failed badly). A failure that comes from the init should repeat all along the row for that init, while a failure that comes from the data ordering should scatter.
- Schedule sweep: peak LR {0.10, 0.07, 0.05, 0.03} × regularizer anneal {on, off}, 32 seeds per condition, trained with the fallback term and scored by the `redirect` intervention (the ex-2.9.2 recipe, so the score reflects anchoring quality).

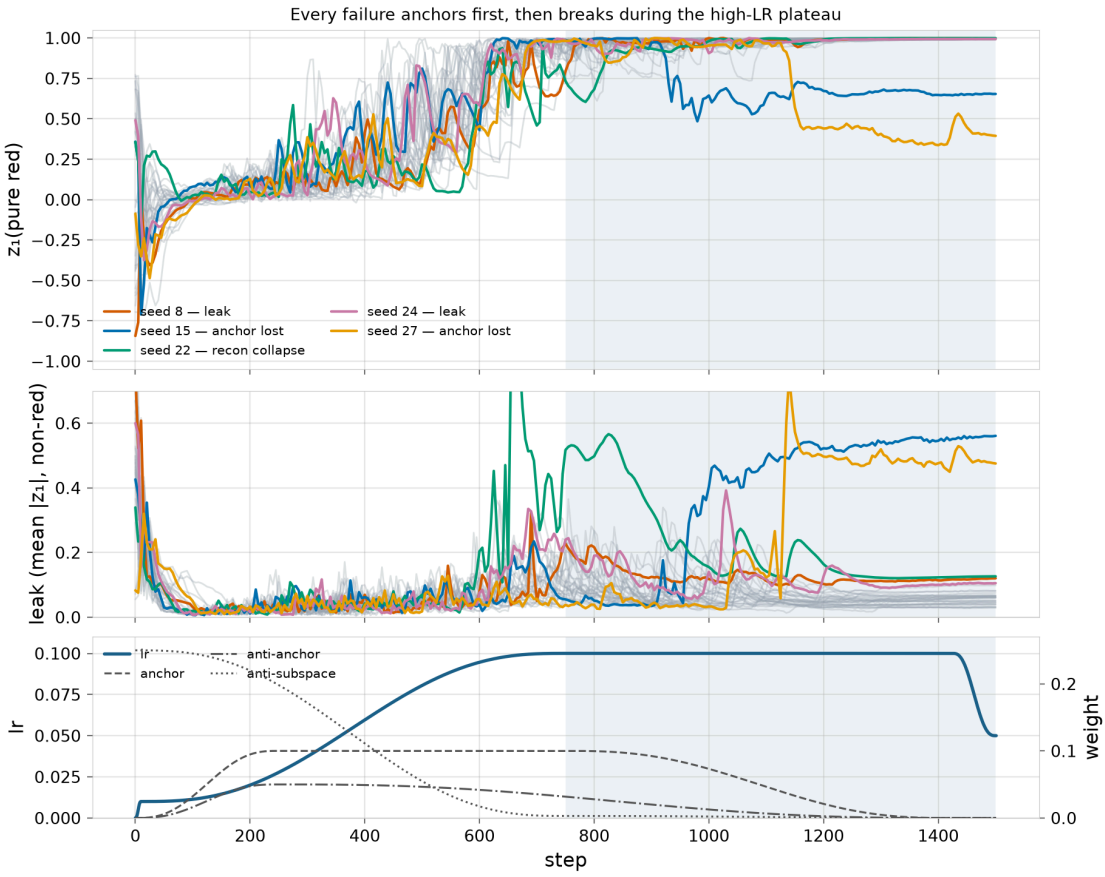
The experiment is `experiment.py`:

```
bin/mini run docs/m1/ex-2.9.3/experiment.py --app moda
```

416 runs completed. The trajectory arm reproduces the base arm of ex-2.9.2: 5 of 32 runs end unhealthy. 3 of them failed catastrophically, meaning the anchor was lost or reconstruction collapsed, and 2 finished with non-red colors leaking onto the anchored axis. The per-step traces show that every failing run anchored successfully first, then broke during the high-LR plateau. The attribution arm confirms the failures are not a property of the initialization, and the sweep finds a plain schedule fix: halve the LR peak and keep the anneal.

When failures happen

Each line below is the training run of one seed under the original schedule. The top panel tracks anchor progress: z_1 of pure red, which reaches 1 when *red* sits exactly on its anchor. The middle panel tracks leakage, the mean $|z_1|$ over colors that are clearly not red. The bottom panel is the schedule itself: the learning rate ramps to its 0.1 peak at step 750 and holds there, while the regularizer weights anneal to zero by step 1425.

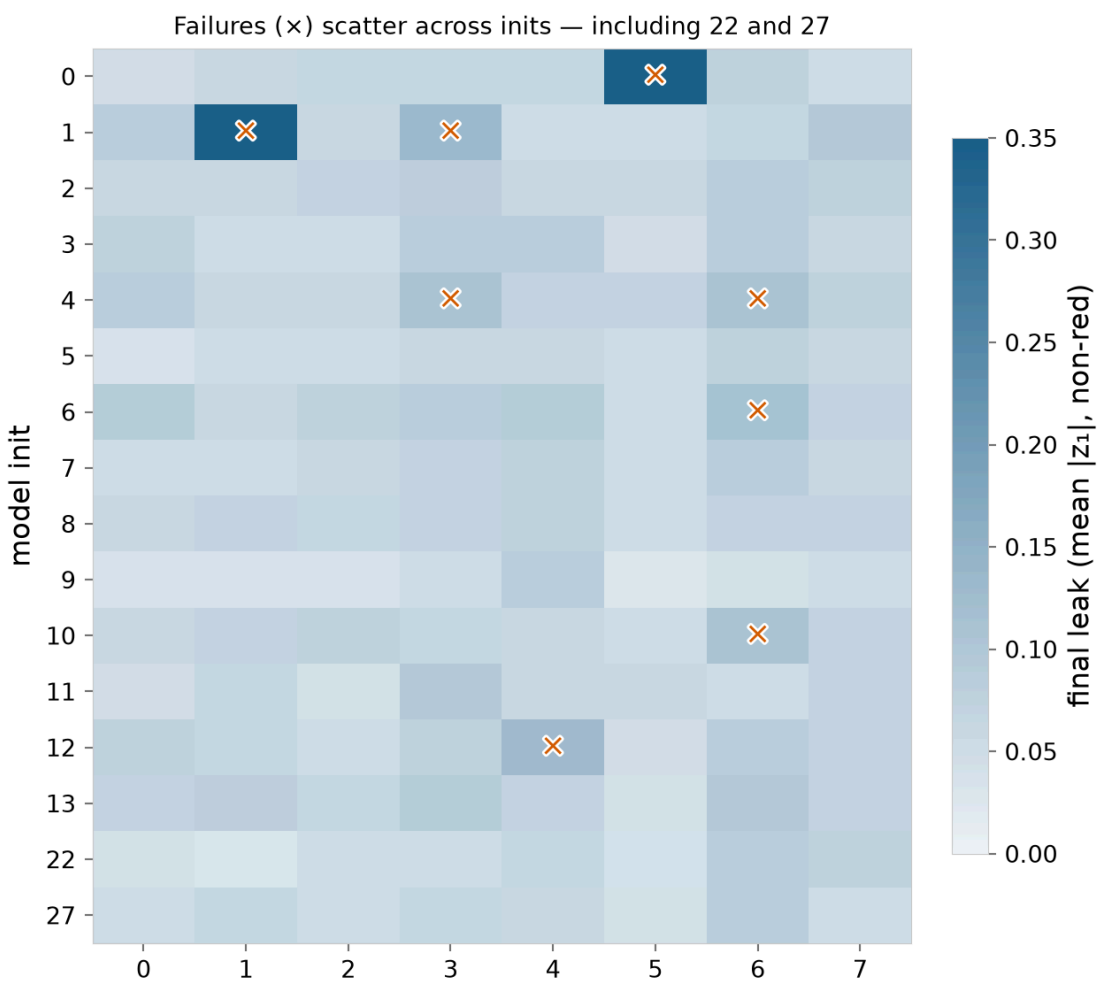


All 5 failing runs reached $z_1(\text{red}) > 0.9$, a clean anchor, between steps 615 and 680, right as the LR approaches its peak. The 4 that later lost it did so between steps 650 and 1145. Seeds 22 and 8 slip the moment the LR tops out, and reconstruction collapses outright on 22. Seeds 15 and 27 slip only once the anchor weight has annealed low: by step 1145 it is 0.034, too weak to pull red back.

So the anchored solution seems to be metastable at this learning rate. The regularizers hold it in place while they are on, and the timed anneal removes that hold while the optimizer is still hot. Nothing about these seeds stopped them from anchoring; they were unlucky during the plateau. If that is right, whether a run fails should follow the randomness of training rather than the initialization, which we test below.

Init versus data stream

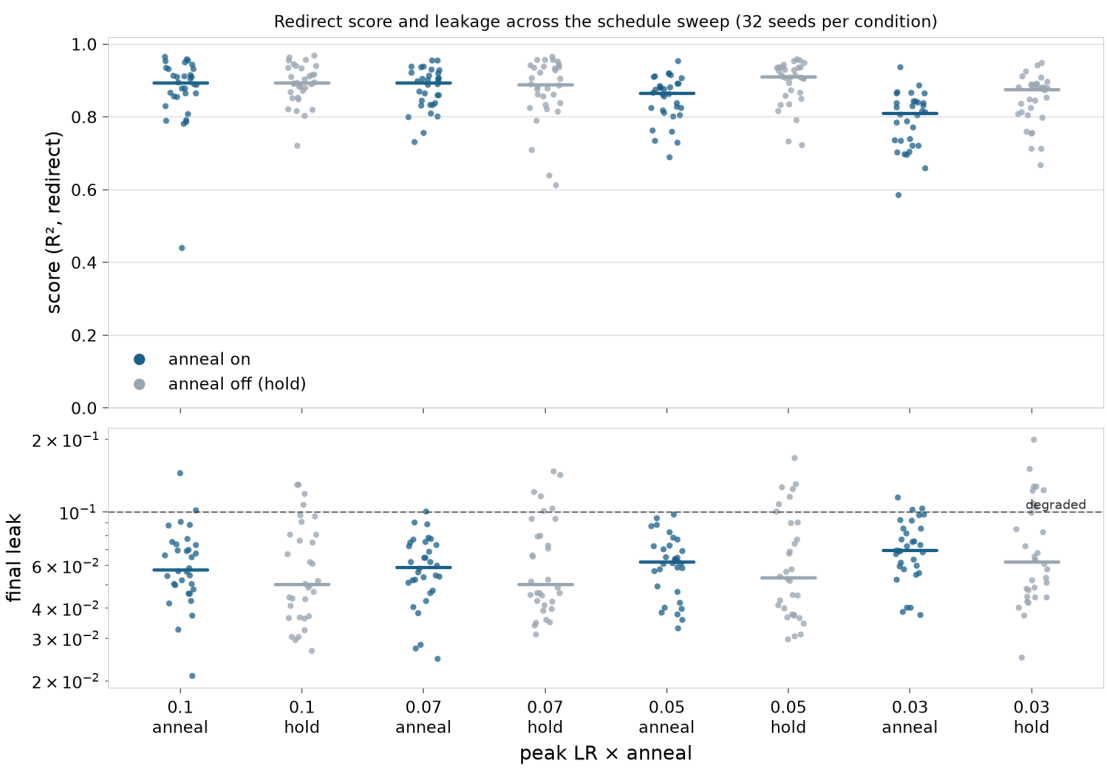
The attribution arm retrains 16 inits × 8 batch/label streams under the original schedule. If failures come from the init, the two inits whose earlier runs failed catastrophically (22 and 27) should fail again. If they come from the luck of training, they should scatter across the grid instead.



Inits 22 and 27, catastrophic under their earlier streams, trained cleanly under **all 8 fresh streams** (0 failures between them). The 8 unhealthy runs (of 128; 2 catastrophic) instead scatter across 6 different inits, none failing more than 2 of 8, with mild clustering by stream (stream 6 accounts for 3). So the incompatible-init hypothesis does not hold: the same init succeeds or fails depending on which random batches and label draws it sees during the hot phase.

Learning rate versus regularizer anneal

At first the anneal looked responsible, since it removes the hold the anchor term provides. But the sweep says otherwise: holding the regularizers on to the end (`anneal off`) makes things worse at every peak, while halving the LR peak removes the failures entirely. These runs train with the fallback term and are scored by the redirect intervention, as in ex-2.9.2.



Three results come out of the sweep.

First, peak LR 0.05 with the anneal is the safe condition: 0/32 unhealthy runs (0.10 gives 2; 0.07 gives 1), a median score of 0.87, and reconstruction as good as the hot condition (median 0.000012 versus 0.000016). On the other hand, at 0.03 the model doesn't train properly, with 3 leaky runs and a median score of 0.81.

Second, the anneal should be kept. Holding the regularizers on lowers typical leak a little, but it widens the tail: 25 leaky runs across the four hold conditions versus 6 with the anneal, because the live anchor term keeps pulling pinkish-labeled samples onto the axis in unlucky runs. Holding on also costs about 10× in reconstruction (median 0.000123 versus 0.000013).

Third, γ should be calibrated per model. The worst hot-condition score (0.44, seed 17) anchored cleanly, but the fixed $\gamma = 1$ redirect bias was too small to dominate the pre-norm scale of that seed, so "deleted" red passed through almost untouched (damage to pure red 0.002). Excluding it, the floor for the hot condition is 0.78.

The fallback term looks like a stabilizer

Catastrophic failures — meaning the anchor was lost, reconstruction collapsed, or leak went beyond 0.3 — occurred only in the fallback-free arms: 5 of 160 base-config runs, against **0 of 256** fallback-trained runs across all eight schedules (0 of 32 in the like-for-like condition, the same schedule as the base arms). Ex-2.9.2 saw this pattern at 32 seeds and was careful about reading much into it, since one extra loss term is hard to tell apart from luck at that scale. At 256 runs the pattern has held without exception.

A mechanism seems plausible. The fallback term pins the decoder output at $-e_1$, which flattens the loss landscape around the redirect direction and may remove the direction along which the instability grows. Only the like-for-like condition is a clean comparison, so the schedule confound prevents this from being conclusive. Still, this is now the second experiment in which fallback training produced zero catastrophes.

Conclusion

The "schedule incompatible with some seeds" hypothesis does not hold up under either test. Failures come late: every failing run anchored first and broke only during the 0.1-LR plateau. And they follow the data stream rather than the init: the earlier bad inits trained cleanly under fresh streams, and the failures scatter. The anchored solution is metastable when the optimizer is too hot, and whether a run stays clean comes down to the luck of the batch stream.

For M2 defaults:

- Reduce the LR. That condition had 0/32 unhealthy runs, scores 0.69 to 0.95 (median 0.87), and reconstruction as good as the hot schedule. Reducing it only during the anneal phase may be enough.
- Keep the fallback term, both for its designed purpose (a predictable intervention response) and for its apparent side effect of preventing catastrophic training failures.
- Calibrate the redirect γ against the pre-norm activation scale of the model. A fixed $\gamma = 1$ silently no-ops on about 1 run in 250.
- Keep the cheap endpoint screening (leak < 0.1, anchor loss, recon). Even the safe condition only bounds what we measured, and the failure mechanism is chaotic.

A tuned schedule fixes the symptom but still requires tuning. Ex-2.9.4 asks whether the weights can instead respond to the training signals themselves, which would remove the timing coupling altogether.