



CONFERENCE

—NORTH AMERICA 2026—

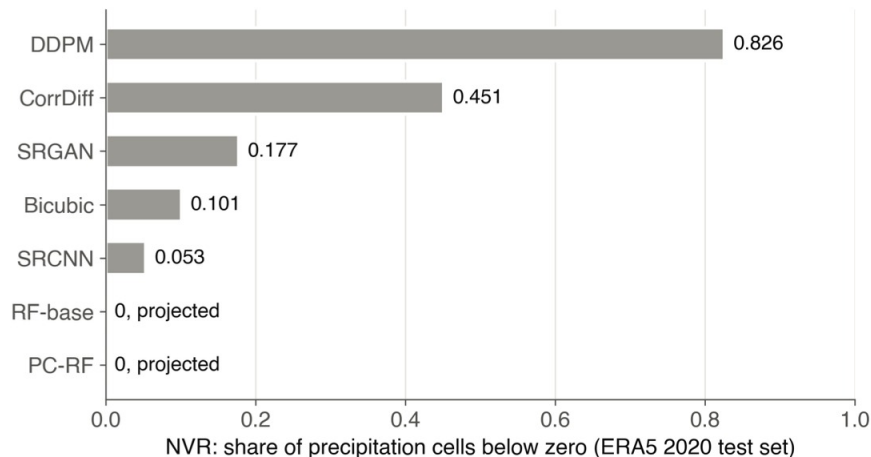
From PyTorch to Production

Serving a physics-constrained generative model with ONNX, Ray, and vLLM

Arun Sharma, University of Minnesota, Twin Cities

Sharp fields can rain below zero

- Our CorrDiff reimplementation has the best PSNR on ERA5 2020, 28.38.
- Yet 45% of its ensemble-mean rain cells are negative.
- Negative rain has no physical meaning in a water balance.



45%

negative ensemble-mean rain cells
CorrDiff (ours), ERA5 2020

Guarantees go in the serving path

Use the training loss to shape the field.

Use a projection at serving time to certify it.

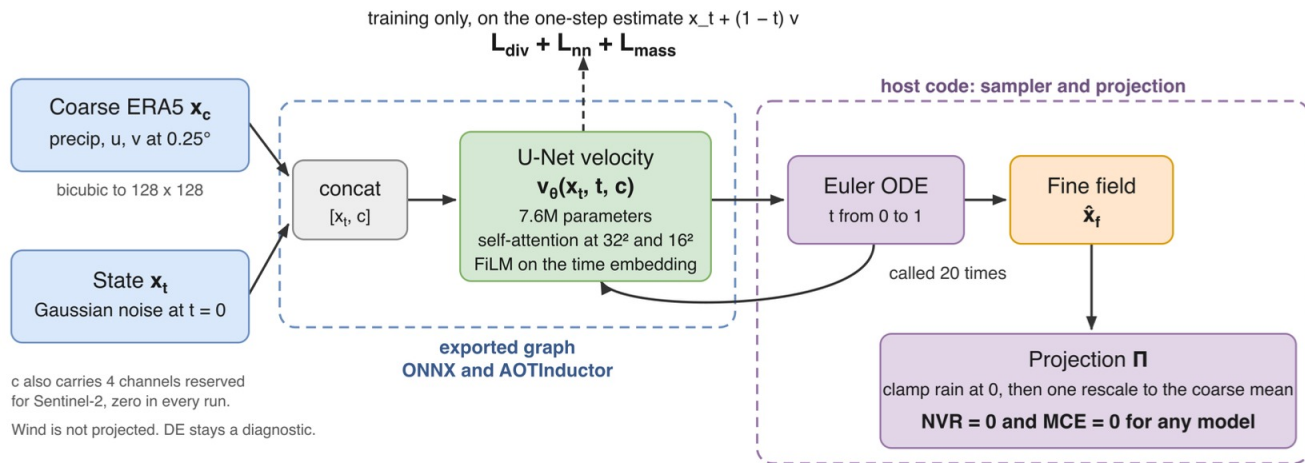
- The projection makes rain non-negative and matches the coarse mean.
- It runs in host code, outside the exported graph.

Task, data, and two corrections

- The input is ERA5 precipitation and 10 m wind at 0.25 degrees.
- The target is ERA5-Land precipitation at 0.1 degrees.
- **Correction:** the target grid is about 8 to 11 km.
- **Correction:** the Sentinel-2 channels were zero in every run.

The model: a 7.6M-parameter U-Net

- Rectified flow trains it to predict a straight-line velocity.
- The dashed blue region is exported, and purple is host code.
- Each of eight ensemble members runs twenty Euler steps.



Physics penalties are just autograd

Simplified from `src/pcrf/physics.py` and `flow.py`

```
x1_hat = x_t + (1 - t) * v      # one-step clean estimate
div = ddx(u) + ddy(v_wind)      # central differences
L_div  = (div ** 2).mean()
L_nn   = (p.clamp(max=0) ** 2).mean()
L_mass = ((p.mean((1,2,3)) - p_coarse_mean) ** 2).mean()
loss = L_rf + L_div + L_nn + L_mass
```

- The penalties act on a one-step clean-field estimate.
- They target divergence, negative rain, and drift from the coarse mean.
- All three weights are fixed at 1.0.
- On ERA5 a sign bug made the divergence term a deformation instead.

The projection is two lines of host code

Simplified from `src/pcrf/physics.py`, `project_physics`

```
p = x[:, 0:1].clamp(min=0.0)
p = p * (coarse_mean / p.mean((1,2,3), keepdim=True))
```

- Clamp rain at zero, then apply one rescale per sample.
- NVR and MCE drop to zero, up to float32 rounding.
- Wind is not projected, so divergence stays a diagnostic.

The domain-level rescaling follows Hess et al., *Nature Machine Intelligence*, 2022.

Export the network, not the sampler

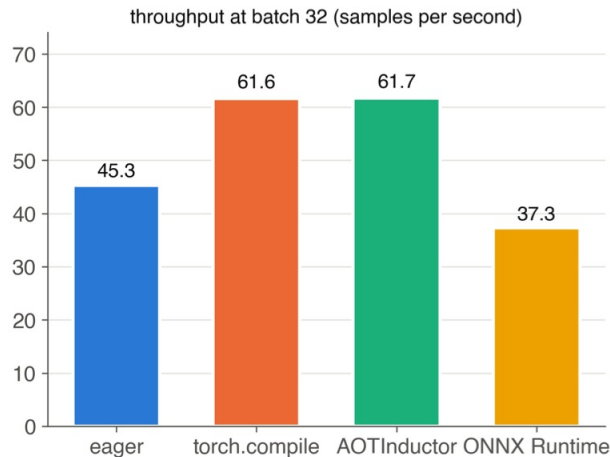
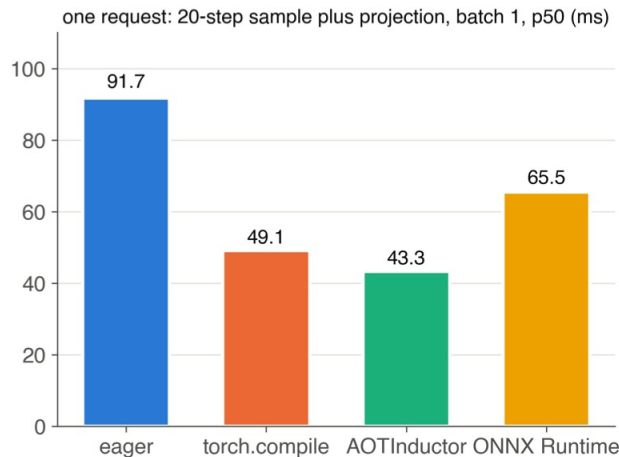
- The exported graph is one velocity call, $v(x_t, t, c)$.
- The Euler loop and the projection stay in host code.
- One graph serves any step count or solver.
- Python and Rust mirror one projection spec, tested separately.

Two export paths, two shape traps

- Legacy torch.onnx froze attention's sequence length at 128 x 128.
- A batch-1 example froze the batch dimension in torch.export.
- A batch-2 example and dynamic batch Dim, 1 to 4096, fix it.
- With TF32 off, AOTInductor matches eager within 2e-6.

What serving costs on one A100

- AOTInductor serves one request in 43.3 ms, eager in 91.7.
- torch.compile takes 49.1 ms, and ONNX Runtime 65.5 ms.
- The weights are random in the trained architecture.



43.3 ms

AOTInductor, one request

91.7 ms

eager, one request

The polyglot path, with honest scope

Component	Status
Host sampler plus projection	● Mirrored in Python and Rust
Rust ONNX Runtime service	● 5 of 5 tests on a demo export
Ray Serve	● CPU smoke on the demo export
Go gateway with Temporal	○ Unit tests with mocked activities
vLLM agent calling the tool	○ Tested against a mock only

● Ran on a demo export or in host code ○ Tested against mocks only

vLLM serves the language model inside the agent, not PC-RF.

Our scaling curve hid a DDP bug

Simplified from the pre-fix train_ray.py, lines 202-203 and 277

```
rf = prepare_model(rf)          # Ray wraps the module in DDP
core = rf.module
loss = core.training_loss(b) # bypasses DDP.forward
# fix: give the module a forward() that returns the loss,
# then call the wrapper: loss = rf(b)
```

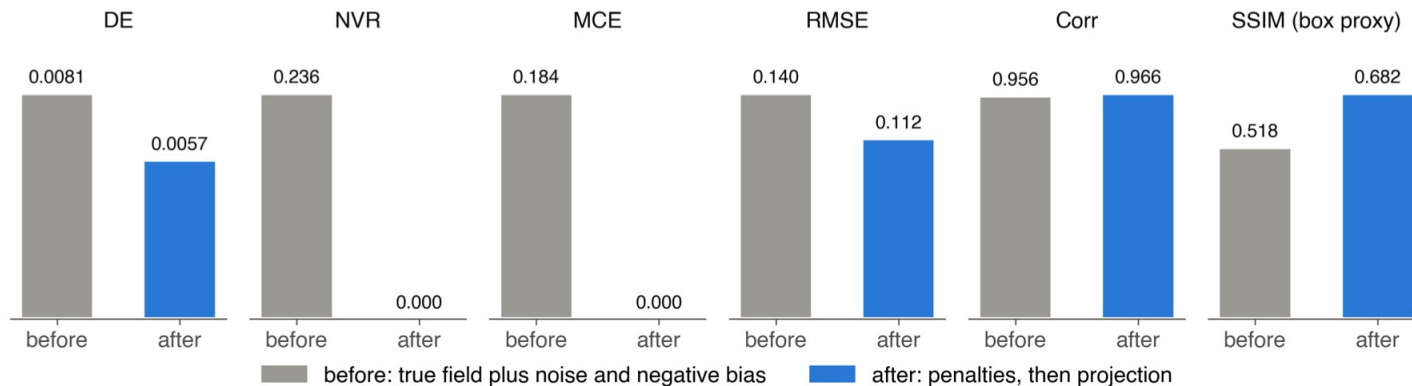
- DDP syncs gradients only through the wrapper's forward call.
- Our earlier curve timed replicas that never synced.
- We fixed it and verified two ranks match after one step.
- A rerun on 4 GPUs gives efficiency 1.00, 0.96, 0.90 at W=1,2,4.

0.90

efficiency at W=4
rerun on 4 GPUs

Synthetic check: the mechanism works

- One constructed 128 x 128 field has exact physics.
- No network is trained in this check.
- The projection drives NVR and MCE to zero.
- The penalties cut divergence error by about 30%.



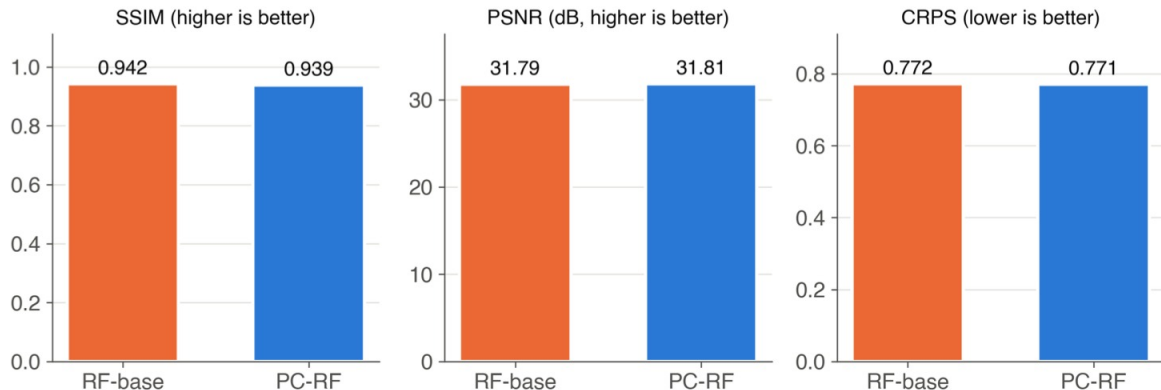
ERA5 2020: only projected models pass

Model	PSNR	CRPS	NVR
Bicubic	26.19	1.356	0.101
CorrDiff (ours)	28.38	1.092	0.451
RF-base (200 epochs)	26.39	1.188	0
PC-RF (300 epochs)	26.59	1.149	0

- RF-base and PC-RF had different budgets, so this pair is unmatched.
- PSNR and CRPS average the rain channel with two wind channels.
- CorrDiff (ours) could apply the projection too, and we never tested that.

Three years, matched budget: a tie

- Every arm trained 100 epochs on 2019–2021.
- RF-base scores SSIM 0.942, and PC-RF scores 0.939.
- The penalties buy no measurable quality here.
- The guarantee belongs to the projection.



SSIM at a matched budget

0.942

■ RF-base, one training seed

0.939

■ PC-RF, one training seed

What the checks certify

- NVR and MCE need only the output and the coarse input.
- Run them first, as a cheap pass-or-fail gate.
- On ERA5, DE measures a deformation term, not divergence.
- The reference wind itself scores 0.0648 on that term.
- Passing is necessary, not sufficient.

Limitations I raise myself

- The reference is interpolated ERA5 over different accumulation windows.
- The enforced mass is the coarse mean over land and sea.
- We used one domain, one test season, and one seed per arm.
- No baseline was projected. ERA5 DE measures deformation, not divergence.
- Serving numbers use random weights of the trained architecture.

Takeaways for PyTorch engineers

- (1) Put exact constraints in host code after the exported graph.
- (2) Export the velocity call, and keep the solver outside.
- (3) Test export parity with TF32 off, at unseen batch sizes.
- (4) Check that DDP ranks hold equal gradients.



CONFERENCE

—NORTH AMERICA 2026—

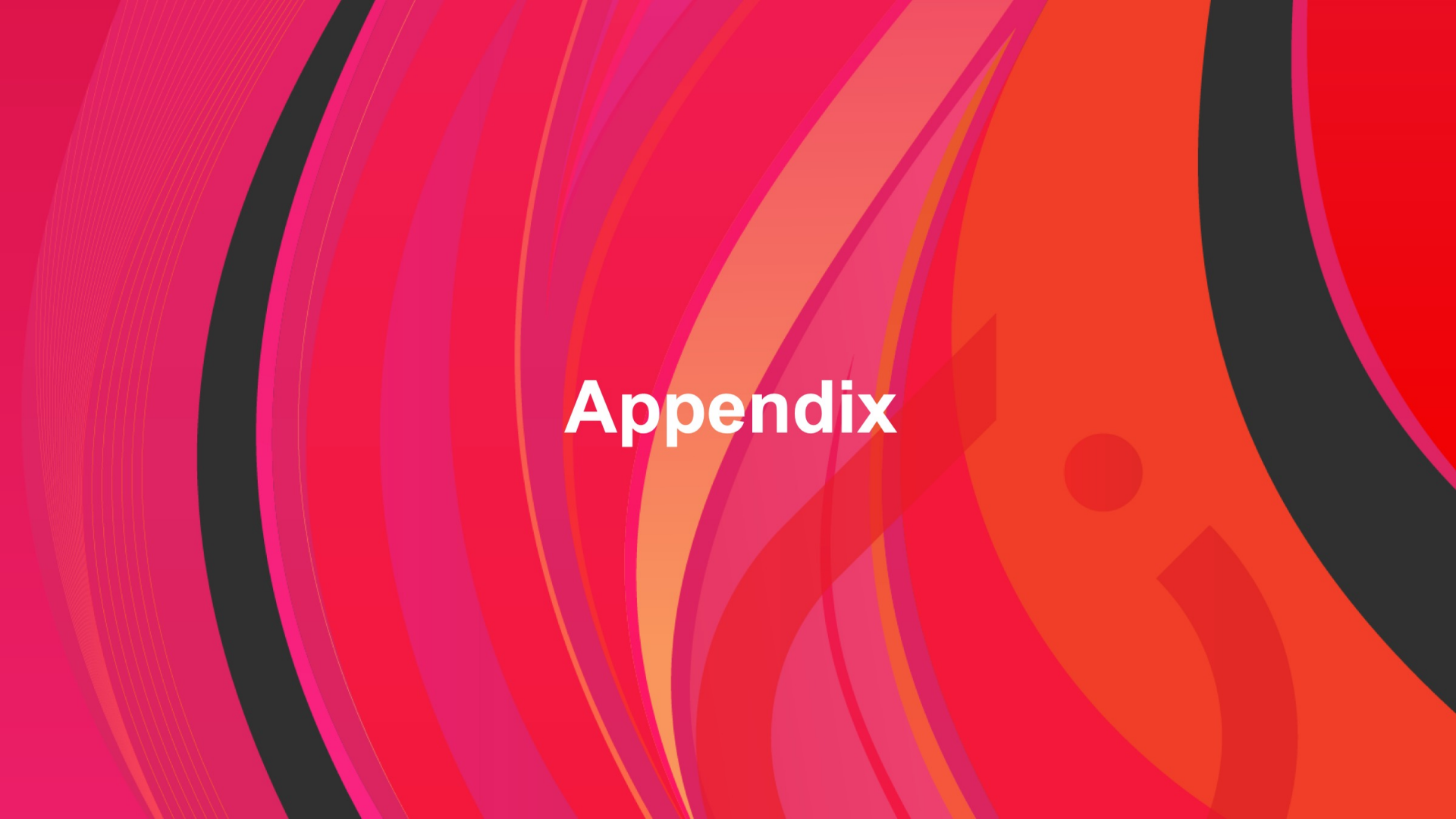
Thank you

Thank you for listening

Arun Sharma, University of Minnesota, Twin Cities
arunshar@umn.edu



Rate this session
sfeedback.com/NN5gQz



Appendix

Training setup

- The main run used 300 epochs, batch 32, AdamW, and lr $2e-4$.
- It had 1171 training samples and 36 steps per epoch.
- One epoch took 74.6 s on an A100-40GB.
- The main run took about 6.2 GPU-hours.
- Main run plus ablations came to about 21 GPU-hours, estimated.

One-year ablation at a matched budget

Arm	SSIM	PSNR	CRPS	DE
Full PC-RF	0.912	26.54	1.161	0.0634
No divergence penalty	0.912	26.58	1.146	0.0649
No non-negativity penalty	0.913	26.55	1.160	0.0644
No mass penalty	0.915	26.50	1.166	0.0639
RF-base	0.913	26.40	1.186	0.0643

Each arm ran with one training seed. On ERA5, DE measures a deformation term, not divergence.

Three-year results

Model	SSIM	PSNR	CRPS	NVR
Bicubic	0.927	31.48	0.873	0.119
CorrDiff (ours)	0.820	32.02	0.761	0.108
RF-base	0.942	31.79	0.772	0
PC-RF	0.939	31.81	0.771	0

Every arm trained 100 epochs on 2019–2021, with one seed per arm.

Ray Train scaling, verified

- The run used $W=1,2,4$ on one A100 node, four GPUs total.
- Each point is real data-parallel training, after the DDP fix.
- Efficiency is 1.00, 0.96, and 0.90 at $W=1, 2, 4$.
- GPU utilization rises with W , which confirms real use.

4-GPU fallback, one a100-4 node, DDP fix verified (job 1736549)

