Diffusion Policy Engine (DPE): Reward-Guided Diffusion for Goal-Directed Sampling

Author: freeky78

Date: October 2025

Abstract. We present a minimal, working prototype of a Diffusion Policy Engine (DPE), which reframes diffusion sampling as a stochastic policy update guided by an external reward. Instead of purely matching data likelihood, the denoising process is augmented with a reward term, nudging reconstructions toward higher utility under a user-defined criterion R(x, p). We demonstrate stable training on a differentiable benchmark (context-shifted Rosenbrock) and outline how the same loop supports black-box rewards via MPPI at inference. The approach is practical: it trains like a standard conditional diffusion model and produces a distribution of high-reward candidates.

1. Method (Plain Formulation)

We generate parameters p conditioned on context x with a conditional denoiser that predicts noise eps_hat(x, p_t, t). We reconstruct p0_hat from the noisy sample and add a reward-guided term to the denoising loss:

```
loss = MSE(eps_hat, eps) - lambda * R(x, p0_hat)
```

At inference, we optionally use a lightweight MPPI reweighting step around the DDPM mean to favor higher-reward candidates. Intuitively, each diffusion step behaves like a stochastic policy update:
`p_{t+1} = p_t + alpha * grad_p E[R(x, p_t)] + sigma_t * eps_t`

2. Minimal Training Snippet (PyTorch)

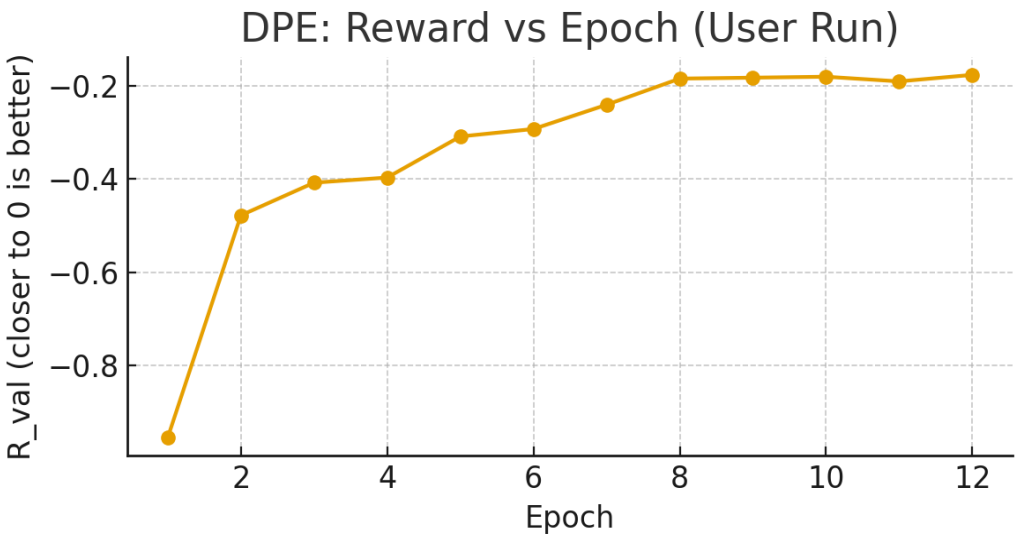
```
eps_hat = model(x, p_t, t01) loss_denoise = F.mse_loss(eps_hat, noise) #
reconstruct p0 from p_t and eps_hat ab = diff.alphas_cumprod[t].view(-1,1)
p0_hat = (p_t - (1-ab).sqrt()*eps_hat) / ab.sqrt() # reward term (squashed for
stability) R = torch.tanh(reward_fn(x, p0_hat) / 5.0) loss = loss_denoise -
lambda_R * R.mean()
```

3. Implementation Highlights

- Stable reward: tanh-squash to bound outliers; warm-up then ramp lambda.
- EMA on reconstructed p0 for smoother reward gradients.
- Optional MPPI at inference (few jittered candidates reweighted by exp(R/lambda)).
- All branches produce consistent hidden dims for clean concatenation.

4. Results (User Run)

Training for 12 epochs with MPPI=4 improved validation reward from about -0.95 to about -0.18 (closer to 0 is better; the benchmark optimum is near 0). Demo contexts produced near-optimal p values, confirming that the DPE loop steers diffusion toward higher utility.



5. Reproduction (CLI)

```
pip install torch>=2.2 numpy pandas matplotlib python dpe_main.py --epochs 12
--mppi_K 4 python make_plot.py # optional: generates results.png
```

6. From Demo to Real-World

Replace the demo reward with your real criterion R(x, p) (e.g., simulator score, model accuracy, control objective). If R is differentiable, keep the same loss. If it is a black-box, keep the denoising loss and rely more on MPPI at inference (increase candidate count). The system then yields a distribution of high-reward candidates conditioned on x.

7. Limitations and Notes

- For sharp reward landscapes, start with small lambda and enable warm-up.
- If gradients explode early, lower the learning rate (e.g., 1e-4) and increase tanh scaling.
- MPPI adds inference cost; keep candidate count small (4 to 16) for practicality.

8. Project Files and Contact

Repository skeleton: dpe_main.py, make_plot.py, requirements.txt, README.md, LICENSE.

Appendix: Logged Reward by Epoch

Epoch	R_val
1	-0.954
2	-0.478
3	-0.408
4	-0.397
5	-0.309
6	-0.293
7	-0.241
8	-0.185
9	-0.183
10	-0.181
11	-0.191
12	-0.177