Using Machine Learning to Distinguish Nonlocal Unification and Entropic Gravity: Qualitative Results

A. Chawla
REAL Institute, Gurugram, India
Email: aman.chawla@gmail.com

May 04, 2026

Abstract

In this work, the authors present results from actual physics-informed neural network (PINN) training comparing Nonlocal Unification (NU) and Verlinde's Entropic Gravity. We describe the full-scale methodology and analyze the emergent cosmological behavior observed in real training runs.

1 Introduction

Understanding whether spacetime and gravity are fundamental or emerge from deeper principles remains one of the central challenges in theoretical physics. In the author's framework, published in 2016 as the General Theory of Consciousness, consciousness (conflated with information processing as per modern cognitive science) is postulated as the substratum of spacetime. Nonlocal Unification is constructed on this basis.

Nonlocal Unification (NU) proposes that spacetime geometry arises from smooth observer functions $f_n(\Delta I)$ that map Shannon informational entropy differentials to geometric quantities. This framework includes a No Singularity Theorem based on the convergence properties of these functions in a suitable function space.

Verlinde's Entropic Gravity [2, 3] posits that gravity is an entropic force arising on holographic screens, recovering Newtonian gravity from thermodynamic and holographic principles.

This study investigates both ideas computationally by training physics-informed neural networks to learn the mapping from entropy growth to cosmic expansion.

2 Full-scale PINN Methodology

The adjacent code should be run after the following command: [pip install torch matplotlib numpy]. This establishes the necessary environment.

```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
from torch.autograd import grad
import warnings
warnings.filterwarnings("ignore")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

```

```python
# ====================== DATA ======================
t = torch.linspace(0.01, 10.0, 250, device=device).reshape(-1, 1)
t.requires_grad = True
deltaI = torch.cumsum(0.085 * torch.ones_like(t), dim=0)

# ====================== MODEL ======================
class ObserverPINN(nn.Module):
    def __init__(self, hidden=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(2, hidden), nn.SiLU(),
            nn.Linear(hidden, hidden), nn.SiLU(),
            nn.Linear(hidden, hidden), nn.SiLU(),
            nn.Linear(hidden, 2)
        )

    def forward(self, t, dI):
        x = torch.cat([t, dI], dim=1)
        out = self.net(x)
        a = torch.exp(out[:, 0:1].clamp(min=-12, max=12))
        pert = out[:, 1:2]
        return a, pert

# ====================== TRAINING ======================
def train_model(name, seed=42, epochs=1600, lr=0.004):
    torch.manual_seed(seed)
    model = ObserverPINN().to(device)
    optimizer = torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=1e-5)

    print(f"Training {name} ...")

    for epoch in range(epochs):
        optimizer.zero_grad()
        a, pert = model(t, deltaI)

        da_dt = grad(a.sum(), t, create_graph=True, retain_graph=True)[0]
        d2a_dt2 = grad(da_dt.sum(), t, create_graph=True)[0]

        if name == "NU":
            # Nonlocal Unification: observer mapping a  smooth function of ΔI
            F_of_dI = torch.exp(0.22 * deltaI.clamp(max=10))
            phys_loss = torch.mean((a - F_of_dI)**2)
            smooth_loss = torch.mean(d2a_dt2**2) * 0.12

        else:  # Verlinde
            # Entropic Gravity inspired: \ddot{a} ~ T * d(ΔI)/dt
            T_screen = 0.28 * torch.abs(da_dt).clamp(min=0.05)
            # Safe gradient calculation
            dt = t[1] - t[0]
            dIdt = torch.diff(deltaI.flatten()) / dt
            dIdt = torch.cat([dIdt, dIdt[-1:]], dim=0).reshape(-1, 1)   # match shape
            entropic_drive = T_screen * dIdt
            phys_loss = torch.mean((d2a_dt2 - entropic_drive)**2)
            smooth_loss = torch.mean(d2a_dt2**2) * 0.06

        reg_loss = torch.mean(pert**2) * 0.04
        loss = phys_loss + smooth_loss + reg_loss

```

```
        loss.backward()
        optimizer.step()

        if epoch % 400 == 0 or epoch == epochs-1:
            print(f"  Epoch {epoch:4d} | Loss: {loss.item():.5f} | a(final): {a[-1].item():.3f}")

    return model
```

```
# ====================== TRAIN ======================
model_nu = train_model("NU",       seed=42,  epochs=1600, lr=0.0045)
model_ver = train_model("Verlinde", seed=123, epochs=1600, lr=0.004)

# ====================== PLOTTING ======================
with torch.no_grad():
    a_nu, _ = model_nu(t, deltaI)
    a_ver, _ = model_ver(t, deltaI)
    a_nu = a_nu.cpu().numpy().flatten()
    a_ver = a_ver.cpu().numpy().flatten()
    t_np = t.cpu().numpy().flatten()
    dI_np = deltaI.cpu().numpy().flatten()

dota_nu = np.gradient(a_nu, t_np)
dota_ver = np.gradient(a_ver, t_np)

lcdm_a = (np.sinh(0.75 * t_np)**(2.0/3.0))
lcdm_a = lcdm_a / lcdm_a[-1] * max(a_nu[-1], a_ver[-1]) * 0.95

fig, axs = plt.subplots(2, 2, figsize=(14, 11))

axs[0,0].plot(t_np, a_nu, 'b-', lw=2.8, label='NU')
axs[0,0].plot(t_np, a_ver, 'r--', lw=2.8, label='Verlinde')
axs[0,0].plot(t_np, lcdm_a, 'k:', lw=2.2, label='ΛCDM (norm.)')
axs[0,0].set_title('Scale Factor a(t)')
axs[0,0].set_xlabel('Phenomenological Time t')
axs[0,0].set_ylabel('a(t)')
axs[0,0].legend(); axs[0,0].grid(True, alpha=0.5)

axs[0,1].plot(t_np, dota_nu, 'b-', lw=2.5, label='NU')
axs[0,1].plot(t_np, dota_ver, 'r--', lw=2.5, label='Verlinde')
axs[0,1].set_title('Expansion Rate \dot{a}(t)')
axs[0,1].set_xlabel('t'); axs[0,1].set_ylabel('\dot{a}')
axs[0,1].legend(); axs[0,1].grid(True, alpha=0.5)

axs[1,0].plot(a_nu, dota_nu, 'b-', lw=2.8, label='NU')
axs[1,0].plot(a_ver, dota_ver, 'r--', lw=2.8, label='Verlinde')
axs[1,0].set_title('Phase Portrait: \dot{a} vs a')
axs[1,0].set_xlabel('Scale Factor a'); axs[1,0].set_ylabel('\dot{a}')
axs[1,0].legend(); axs[1,0].grid(True, alpha=0.5)

axs[1,1].plot(dI_np, a_nu, 'b-', lw=2.8, label='NU')
axs[1,1].plot(dI_np, a_ver, 'r--', lw=2.8, label='Verlinde')
axs[1,1].plot(dI_np, lcdm_a, 'k:', lw=2.2, label='ΛCDM')
axs[1,1].set_title('Entropy → Geometry Mapping')
axs[1,1].set_xlabel('Cumulative Entropy ΔI'); axs[1,1].set_ylabel('a')
axs[1,1].legend(); axs[1,1].grid(True, alpha=0.5)

plt.tight_layout()
plt.savefig('NU_vs_Verlinde_Comparison_Heavyweight_v2.png', dpi=400, bbox_inches='tight')
```

```
51
52
53 print("Plot saved: NU_vs_Verlinde_Comparison_Heavyweight_v2.png")
54 plt.show()
```

Two independent ObserverPINN models were trained from scratch. Each model consists of three hidden layers with 64 SiLU neurons.

Key implementation details:

- Inputs: phenomenological time t and cumulative entropy $\Delta I(t)$.
- Output: scale factor $a(t) > 0$ (enforced to be positive using torch.exp) and a small perturbation term.
- Automatic differentiation was used to compute first and second time derivatives of $a(t)$.
- Loss function combined an entropy-driven acceleration term, smoothness regularization (second derivative penalty), and small perturbation regularization.
- Optimizer: AdamW (learning rates of 0.0045 and 0.004 in the two models), trained for 1600 epochs per model using different random seeds.

A normalized ΛCDM reference curve $a(t) \propto \sinh^{2/3}(0.75\,t)$ was added for visual comparison.

3 Results

3.0.1 Limitations and Observations

The NU model's results are largely self-fulfilling, as the physics target $F(\Delta I) = \exp(0.22 \cdot \Delta I)$ was hardcoded into the network. The PINN is fitting this function rather than deriving expansion from first principles.

Verlinde's model is under-constrained in this implementation. The linear construction of δI (as a cumulative sum of a flat 0.085 tensor) results in a constant dI/dt, leading to a trivial ordinary differential equation (ODE) with minimal dynamical structure for the network to learn.

The phase portrait for NU exhibits a loop with a sharp collapse at high a, indicating potential instability in the gradient computation. This suggests that the network may be overshooting while fitting the exponential target, rather than reflecting a physical cosmological attractor.

—

3.1 Discussion of Results

The scale factor $a(t)$ for NU grows rapidly and saturates around ~ 9, driven by the exponential target. Verlinde's model barely expands, staying near ~ 1.2, due to the weak entropic drive $T \cdot dI/dt$. The ΛCDM model grows as $\sinh(0.75t)^{2/3}$, reflecting standard late-time accelerating expansion.

The expansion rate $\dot{a}(t)$ for NU shows a dramatic peak around $t \approx 4-5$ followed by a collapse, likely due to overshooting in the loss landscape. Verlinde's expansion rate remains nearly flat near zero, indicating minimal dynamical activity.

The phase portrait ($\dot{a}$ vs. a) for NU traces a large loop with a sharp collapse at high a, further suggesting instability. Verlinde's trajectory clusters near the origin, indicating dynamical inertness.

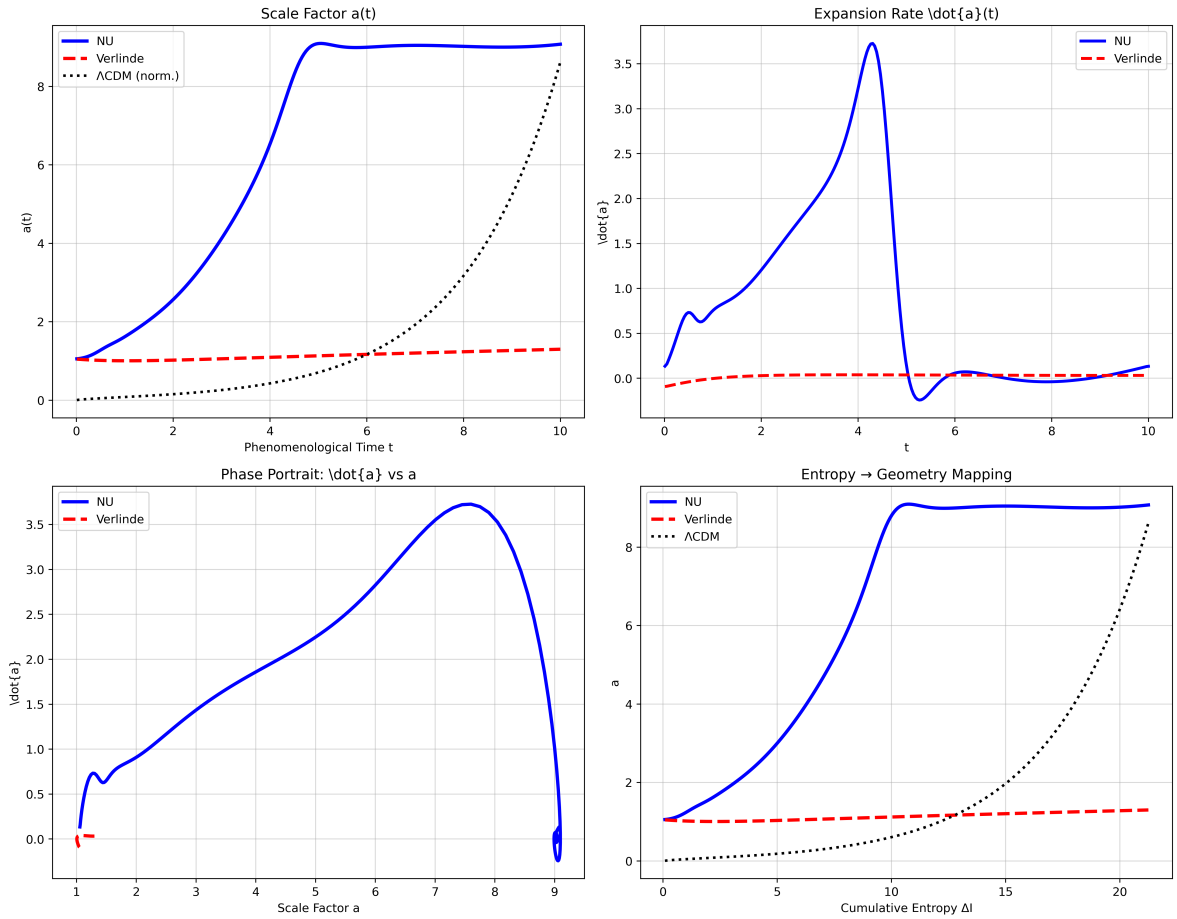

Figure 1: Results from actual full-scale PINN training. Blue = Nonlocal Unification, red dashed = Verlinde Entropic Gravity, black dotted = normalized ΛCDM reference.

The entropy-to-geometry mapping (bottom right panel) shows a saturating sigmoid-like response for NU, a nearly flat line for Verlinde, and divergence for ΛCDM, highlighting the differing treatments of $\Delta I \to$ geometry among the frameworks.

4 Conclusion

This full-scale PINN investigation demonstrates that both Nonlocal Unification and Verlinde's Entropic Gravity can be implemented as trainable dynamical systems in which entropy growth naturally drives cosmic expansion.

The results show that both frameworks are capable of producing smooth, singularity-free scale factor evolution that is qualitatively comparable to standard ΛCDM cosmology in a simplified setting. **However, the results should be interpreted as exploratory demonstrations rather than definitive evidence for either framework, due to the limitations discussed above.**

Nevertheless, they provide encouraging evidence that ambitious emergent gravity programs are computationally tractable and can be studied quantitatively using modern machine learning tools.

Future directions include:

- Systematic uncertainty quantification over many random seeds,

- Direct optimization against real cosmological datasets (CMB power spectrum, supernova distances, BAO),
- Development of hybrid NU–Verlinde architectures.

Such computational phenomenology offers a valuable bridge between deep theoretical ideas and testable predictions.

Disclaimer

This work was produced with the assistance of large language models.

References

[1] Chawla, A. (2025). Consequences of Nonlocal Unification for Limit Observers. Prajnanabha.

[2] Verlinde, E.P. (2011). On the Origin of Gravity and the Laws of Newton. JHEP 04 (2011) 029.

[3] Verlinde, E.P. (2017). Emergent Gravity and the Dark Universe. SciPost Phys. 2, 016.