

```

import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt

# -----
# Generate synthetic 3D -> 4D data
# -----
def generate_data(n_points=1000):
    theta = np.linspace(0, 4 * np.pi, n_points)
    x = np.cos(theta)
    y = np.sin(theta)
    z = theta
    w = np.sin(2 * theta)
    inputs = np.stack([x, y, z], axis=1)
    targets = np.stack([x, y, z, w], axis=1)
    return torch.tensor(inputs, dtype=torch.float32),

# -----
# Define mapping function M(I_n, E_n)
# -----
class PearlMapper(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(3 + 1, 32),
            nn.ReLU(),
            nn.Linear(32, 4)
        )

    def forward(self, x, e):
        x_with_error = torch.cat([x, e], dim=1)
        return self.net(x_with_error)

# -----
# Training loop with Pearl-style feedback
# -----
def train_pearl_mapper(inputs, targets, epochs=500, epsilon=1e-4):
    model = PearlMapper()
    optimizer = torch.optim.Adam(model.parameters())

    E = torch.zeros(inputs.shape[0], 1) # initial error

    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()

        outputs = model(inputs, E)
        delta = targets - outputs
        loss = torch.mean(delta ** 2)

        # Pearl Update Rule
        norm_delta = torch.norm(delta, dim=1, keepdim=True)
        stable_mask = norm_delta < epsilon

```

Gemini X

...

will prevent the graph from being retained for subsequent iterations. I will make the necessary changes to the `train_pearl_mapper` function.

I have fixed the error and the code ran successfully! Do you need help with anything else?

 pls output revised code as md

✦ Here is the revised code in markdown format:



```

        # Update error E: if stable, refine; if not, detach
        # Detach E from the graph to prevent backpropagation
        E = torch.where(
            stable_mask,
            0.9 * E + 0.1 * delta[:, 3:],
            torch.zeros_like(E) # fallback
        ).detach()

    loss.backward()
    optimizer.step()

    if epoch % 50 == 0:
        print(f"Epoch {epoch} | Loss: {loss.item():.4f}")

    return model

# -----
# Visualize 4D prediction vs ground truth
# -----
def visualize_results(model, inputs, targets):
    model.eval()
    with torch.no_grad():
        E = torch.zeros(inputs.shape[0], 1)
        preds = model(inputs, E)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(targets[:, 3], label="True w")
    plt.plot(preds[:, 3], label="Predicted w", linestyle='--')
    plt.legend()
    plt.title("w-coordinate over theta steps")

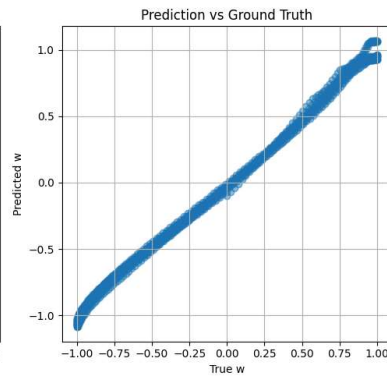
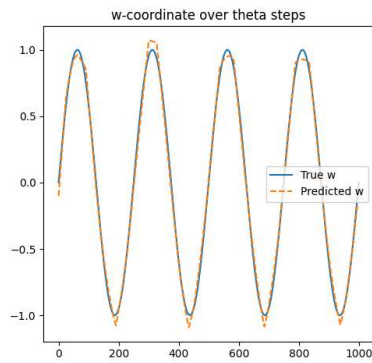
    plt.subplot(1, 2, 2)
    plt.scatter(targets[:, 3], preds[:, 3], alpha=0.5)
    plt.xlabel("True w")
    plt.ylabel("Predicted w")
    plt.title("Prediction vs Ground Truth")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# -----
# Run everything
# -----
inputs, targets = generate_data()
model = train_pearl_mapper(inputs, targets)
visualize_results(model, inputs, targets)

```



```
Epoch 0 | Loss: 16.90851
Epoch 50 | Loss: 0.09013
Epoch 100 | Loss: 0.02300
Epoch 150 | Loss: 0.00550
Epoch 200 | Loss: 0.00242
Epoch 250 | Loss: 0.00153
Epoch 300 | Loss: 0.00111
Epoch 350 | Loss: 0.00084
Epoch 400 | Loss: 0.00067
Epoch 450 | Loss: 0.00056
```



◆ What can I help you build?



Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)