```python
# ================================
# STEP 1: IMPORT LIBRARIES
# ================================
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ================================
# STEP 2: DATASET PREPARATION
# ================================
transform = transforms.ToTensor()

train_dataset = datasets.MNIST(
    root="./data",
    train=True,
    download=True,
    transform=transform
)

test_dataset = datasets.MNIST(
    root="./data",
    train=False,
    download=True,
    transform=transform
)

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 85.1MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 23.8MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 89.5MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 5.32MB/s]
```

```python
# ================================
# STEP 3–5: VAE MODEL (FIXED)
# ================================
class VAE(nn.Module):
    def __init__(self, latent_dim=20):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(28 * 28, 400)
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)

        self.fc2 = nn.Linear(latent_dim, 400)
```

```
        self.fc3 = nn.Linear(400, 28 * 28)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        h = torch.relu(self.fc1(x))

        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)

        z = self.reparameterize(mu, logvar)

        h_dec = torch.relu(self.fc2(z))
        x_recon = torch.sigmoid(self.fc3(h_dec))

        return x_recon, mu, logvar

# ================================
# STEP 6: LOSS FUNCTION
# ================================
def vae_loss(recon_x, x, mu, logvar):
    recon_x = recon_x.view(-1, 28 * 28)
    x = x.view(-1, 28 * 28)

    bce = nn.functional.binary_cross_entropy(recon_x, x,
reduction="sum")
    kl = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return bce + kl

# ================================
# STEP 7: INITIALIZATION
# ================================
model = VAE(latent_dim=20).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)

epochs = 10
losses = []

# ================================
# STEP 8: TRAINING
# ================================
for epoch in range(epochs):
    model.train()
    total_loss = 0

    for x, _ in train_loader:
```

```
        x = x.to(device)

        optimizer.zero_grad()
        recon_x, mu, logvar = model(x)
        loss = vae_loss(recon_x, x, mu, logvar)

        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(train_loader.dataset)
    losses.append(avg_loss)

    print(f"Epoch [{epoch+1}/{epochs}] Loss: {avg_loss:.2f}")
```

```
Epoch [1/10] Loss: 165.23
Epoch [2/10] Loss: 122.18
Epoch [3/10] Loss: 114.85
Epoch [4/10] Loss: 111.78
Epoch [5/10] Loss: 110.05
Epoch [6/10] Loss: 108.77
Epoch [7/10] Loss: 107.95
Epoch [8/10] Loss: 107.32
Epoch [9/10] Loss: 106.82
Epoch [10/10] Loss: 106.37
```

```
# ================================
# STEP 9: RECONSTRUCTION
# ================================
model.eval()
with torch.no_grad():
    x, _ = next(iter(test_loader))
    x = x.to(device)
    recon, _, _ = model(x)

plt.figure(figsize=(8, 4))
for i in range(8):
    plt.subplot(2, 8, i + 1)
    plt.imshow(x[i].cpu().view(28, 28), cmap="gray")
    plt.axis("off")

    plt.subplot(2, 8, i + 9)
    plt.imshow(recon[i].cpu().view(28, 28), cmap="gray")
    plt.axis("off")

plt.show()
```

```
# ================================
# STEP 10: GENERATION
# ================================
with torch.no_grad():
    z = torch.randn(16, 20).to(device)
    samples = model.fc3(torch.relu(model.fc2(z)))
    samples = samples.view(-1, 28, 28)

plt.figure(figsize=(4, 4))
for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.imshow(samples[i].cpu(), cmap="gray")
    plt.axis("off")

plt.show()
```

```
# ================================
# STEP 11: LOSS CURVE
# ================================
plt.plot(losses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("VAE Training Loss")
plt.show()
```