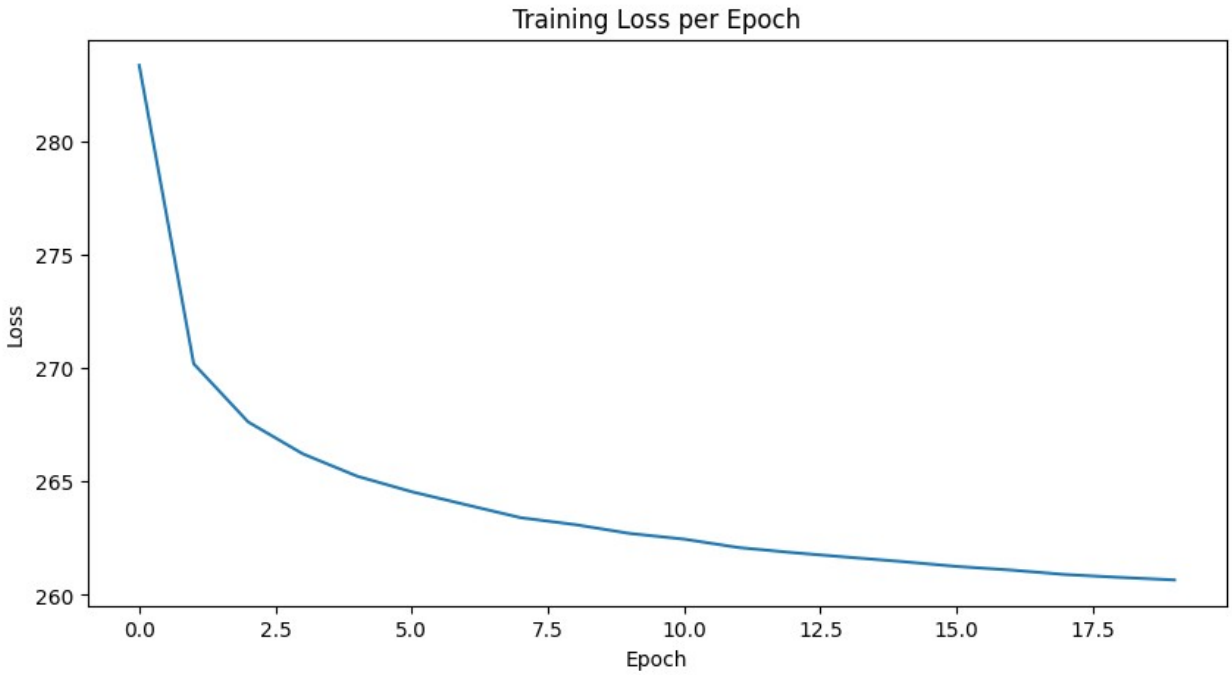
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np

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

BATCH_SIZE = 64
LEARNING_RATE = 1e-3
EPOCHS = 20
LATENT_DIM = 2

# Task 1: Dataset Preparation
transform = transforms.Compose([
    transforms.ToTensor(), # Converts to [0, 1]
])


train_dataset = datasets.FashionMNIST(root='./data', train=True,
transform=transform, download=True)
test_dataset = datasets.FashionMNIST(root='./data', train=False,
transform=transform, download=True)

train_loader = DataLoader(dataset=train_dataset,
batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE,
shuffle=False)

class VAE(nn.Module):
    def __init__(self, latent_dim=2):
        super(VAE, self).__init__()

        # --- Encoder Network
        self.fc1 = nn.Linear(784, 400)
        self.relu = nn.ReLU()

        # Output two vectors: Mean (mu) and Log-Variance (log_var)
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)

        # --- Decoder Network  ---
        self.fc3 = nn.Linear(latent_dim, 400)
        self.fc4 = nn.Linear(400, 784)
        self.sigmoid = nn.Sigmoid() # Squish output between [0, 1] for
image pixels
```

```
    def encode(self, x):
        h1 = self.relu(self.fc1(x))
        return self.fc_mu(h1), self.fc_logvar(h1)

    def reparameterize(self, mu, logvar):
        # Task 2: Reparameterization Trick
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h3 = self.relu(self.fc3(z))
        return self.sigmoid(self.fc4(h3))
    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

model = VAE(latent_dim=LATENT_DIM).to(device)
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)


# Task 3: Define Loss Function [cite: 32]
def loss_function(recon_x, x, mu, logvar):
    BCE = nn.functional.binary_cross_entropy(recon_x, x.view(-1, 784),
reduction='sum')

    # 2. KL Divergence Loss
    # Formula: -0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return BCE + KLD

    # Task 4: Train the VAE [cite: 33]
model.train()
train_losses = []

print("Starting Training...")
for epoch in range(EPOCHS):
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)

        # Forward pass
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)

        # Compute loss
        loss = loss_function(recon_batch, data, mu, logvar)
```

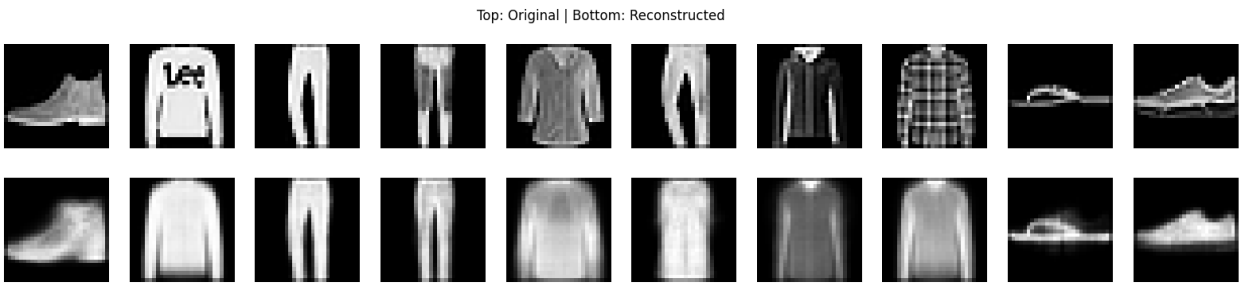
```
        # Backward pass
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    avg_loss = train_loss / len(train_loader.dataset)
    train_losses.append(avg_loss)
    print(f'Epoch: {epoch+1} | Average Loss: {avg_loss:.4f}')
# Plot Loss Curve
plt.figure(figsize=(10,5))
plt.plot(train_losses)
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```

```
Using device: cuda

100%|██████████| 26.4M/26.4M [00:01<00:00, 13.3MB/s]
100%|██████████| 29.5k/29.5k [00:00<00:00, 211kB/s]
100%|██████████| 4.42M/4.42M [00:01<00:00, 3.94MB/s]
100%|██████████| 5.15k/5.15k [00:00<00:00, 13.2MB/s]

Starting Training...
Epoch: 1 | Average Loss: 283.3538
Epoch: 2 | Average Loss: 270.1935
Epoch: 3 | Average Loss: 267.6286
Epoch: 4 | Average Loss: 266.2254
Epoch: 5 | Average Loss: 265.2360
Epoch: 6 | Average Loss: 264.5517
Epoch: 7 | Average Loss: 263.9769
Epoch: 8 | Average Loss: 263.4067
Epoch: 9 | Average Loss: 263.1020
Epoch: 10 | Average Loss: 262.7127
Epoch: 11 | Average Loss: 262.4607
Epoch: 12 | Average Loss: 262.0902
Epoch: 13 | Average Loss: 261.8648
Epoch: 14 | Average Loss: 261.6652
Epoch: 15 | Average Loss: 261.4725
Epoch: 16 | Average Loss: 261.2577
Epoch: 17 | Average Loss: 261.0974
Epoch: 18 | Average Loss: 260.8999
Epoch: 19 | Average Loss: 260.7738
Epoch: 20 | Average Loss: 260.6619
```




```
# Task 5: Sample Generation [cite: 36]
def visualize_results(model, test_loader, num_samples=10):
    model.eval()

    # 1. Reconstruction Check
    data, _ = next(iter(test_loader))
    data = data.to(device)
    with torch.no_grad():
        recon, _, _ = model(data)

    # Plot Original vs Reconstructed
    fig, axes = plt.subplots(2, num_samples, figsize=(20, 4))
    for i in range(num_samples):
        # Original
        axes[0, i].imshow(data[i].cpu().numpy().squeeze(),
cmap='gray')
        axes[0, i].axis('off')
        # Reconstructed
        axes[1, i].imshow(recon[i].cpu().view(28, 28).numpy(),
cmap='gray')
        axes[1, i].axis('off')
    plt.suptitle("Top: Original | Bottom: Reconstructed")
    plt.show()

    # 2. Generate NEW Samples from Random Noise [cite: 37-39]
    with torch.no_grad():
        # Sample random vectors from standard normal distribution
        z = torch.randn(num_samples, LATENT_DIM).to(device)
        generated_images = model.decode(z).cpu().view(num_samples, 28,
```

```
28)

    plt.figure(figsize=(20, 4))
    for i in range(num_samples):
        plt.subplot(1, num_samples, i+1)
        plt.imshow(generated_images[i], cmap='gray')
        plt.axis('off')
    plt.suptitle("Newly Generated Fashion Items (from Noise)")
    plt.show()

# Task 6: Latent Space Visualization (Optional) [cite: 40-42]
def plot_latent_space(model, test_loader):
    model.eval()
    all_z = []
    all_labels = []

    with torch.no_grad():
        for data, labels in test_loader:
            data = data.to(device)
            mu, _ = model.encode(data.view(-1, 784))
            all_z.append(mu.cpu())
            all_labels.append(labels)

    all_z = torch.cat(all_z).numpy()
    all_labels = torch.cat(all_labels).numpy()

    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(all_z[:, 0], all_z[:, 1], c=all_labels,
cmap='tab10', alpha=0.6, s=2)
    plt.colorbar(scatter)
    plt.title("Latent Space Visualization (Fashion MNIST)")
    plt.xlabel("Latent Dim 1")
    plt.ylabel("Latent Dim 2")
    plt.show()

# Run Visualizations
visualize_results(model, test_loader)
plot_latent_space(model, test_loader)
```



Newly Generated Fashion Items (from Noise)