```
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
transform = transforms.Compose([transforms.ToTensor()])


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)

# --- 2. VAE Architecture
class VAE(nn.Module):
    def __init__(self, latent_dim=2):
        super(VAE, self).__init__()
        # Encoder
        self.fc1 = nn.Linear(784, 400)
        self.relu = nn.ReLU()
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)
        # Decoder
        self.fc3 = nn.Linear(latent_dim, 400)
        self.fc4 = nn.Linear(400, 784)
        self.sigmoid = nn.Sigmoid()

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

    def reparameterize(self, mu, logvar):
        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

# --- 3. Training Function
def train_experiment(use_kld=True, epochs=10):
    model = VAE(latent_dim=2).to(device)
    optimizer = optim.Adam(model.parameters(), lr=1e-3)

    bce_history = []

    print(f"\n--- Starting Training: KLD {'ENABLED' if use_kld else
'DISABLED'} ---")
    for epoch in range(epochs):
        total_bce = 0
        model.train()
        for data, _ in train_loader:
            data = data.to(device)
            optimizer.zero_grad()

            recon_batch, mu, logvar = model(data)

            # Reconstruction Loss (Cross Entropy)
            BCE = nn.functional.binary_cross_entropy(recon_batch,
data.view(-1, 784), reduction='sum')

            # KL Divergence [cite: 31]
            if use_kld:
                KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) -
logvar.exp())
            else:
                KLD = torch.tensor(0.0).to(device) # Disable KLD

            loss = BCE + KLD
            loss.backward()
            optimizer.step()

            total_bce += BCE.item()

        avg_bce = total_bce / len(train_loader.dataset)
        bce_history.append(avg_bce)
        print(f"Epoch {epoch+1}/{epochs} | BCE Loss: {avg_bce:.2f}")

    return model, bce_history
```

```python
# 1. Without KL (Standard Autoencoder behavior)
model_no_kl, loss_no_kl = train_experiment(use_kld=False, epochs=10)

# 2. With KL (VAE behavior)
model_with_kl, loss_with_kl = train_experiment(use_kld=True,
epochs=10)
# --- 4. Visualization & Comparison
def compare_latent_space(model_no_kl, model_with_kl, loader):
    model_no_kl.eval()
    model_with_kl.eval()

    z_no_kl, labels_list = [], []
    z_with_kl = []

    with torch.no_grad():
        for data, labels in loader:
            data = data.to(device)
            # Encode data to latent space
            mu_no, _ = model_no_kl.encode(data.view(-1, 784))
            mu_with, _ = model_with_kl.encode(data.view(-1, 784))

            z_no_kl.append(mu_no.cpu())
            z_with_kl.append(mu_with.cpu())
            labels_list.append(labels)

            if len(z_no_kl) * BATCH_SIZE > 2000: break

    z_no_kl = torch.cat(z_no_kl).numpy()
    z_with_kl = torch.cat(z_with_kl).numpy()
    labels = torch.cat(labels_list).numpy()

    # Plotting
    fig, axes = plt.subplots(1, 2, figsize=(14, 6))

    # Plot No KL
    scatter1 = axes[0].scatter(z_no_kl[:, 0], z_no_kl[:, 1], c=labels,
cmap='tab10', s=5, alpha=0.7)
    axes[0].set_title("Latent Space WITHOUT KL (Messy)")
    axes[0].set_xlabel("Latent Dim 1")
    axes[0].set_ylabel("Latent Dim 2")
    # Plot With KL
    scatter2 = axes[1].scatter(z_with_kl[:, 0], z_with_kl[:, 1],
c=labels, cmap='tab10', s=5, alpha=0.7)
    axes[1].set_title("Latent Space WITH KL (Structured/Normal)")
    axes[1].set_xlabel("Latent Dim 1")

    plt.colorbar(scatter2, ax=axes.ravel().tolist())
    plt.show()

def compare_generation(model_no_kl, model_with_kl):
```

```
    # Generate from pure noise (Standard Normal)
    z = torch.randn(10, 2).to(device)

    with torch.no_grad():
        gen_no_kl = model_no_kl.decode(z).cpu().view(10, 28, 28)
        gen_with_kl = model_with_kl.decode(z).cpu().view(10, 28, 28)

    fig, axes = plt.subplots(2, 10, figsize=(20, 5))
    for i in range(10):
        # No KL Generation (Usually Garbage)
        axes[0, i].imshow(gen_no_kl[i], cmap='gray')
        axes[0, i].axis('off')
        if i == 5: axes[0, i].set_title("Generated WITHOUT KL",
fontsize=14, color='red')

        # With KL Generation (Valid Clothes)
        axes[1, i].imshow(gen_with_kl[i], cmap='gray')
        axes[1, i].axis('off')
        if i == 5: axes[1, i].set_title("Generated WITH KL",
fontsize=14, color='green')

    plt.show()

# Execute comparisons
compare_latent_space(model_no_kl, model_with_kl, test_loader)
compare_generation(model_no_kl, model_with_kl)

# Print Final Loss Comparison
print(f"Final BCE Loss (Without KL): {loss_no_kl[-1]:.2f} (Focuses
only on memorization)")
print(f"Final BCE Loss (With KL):    {loss_with_kl[-1]:.2f} (Slightly
higher tradeoff for valid generation)")
```

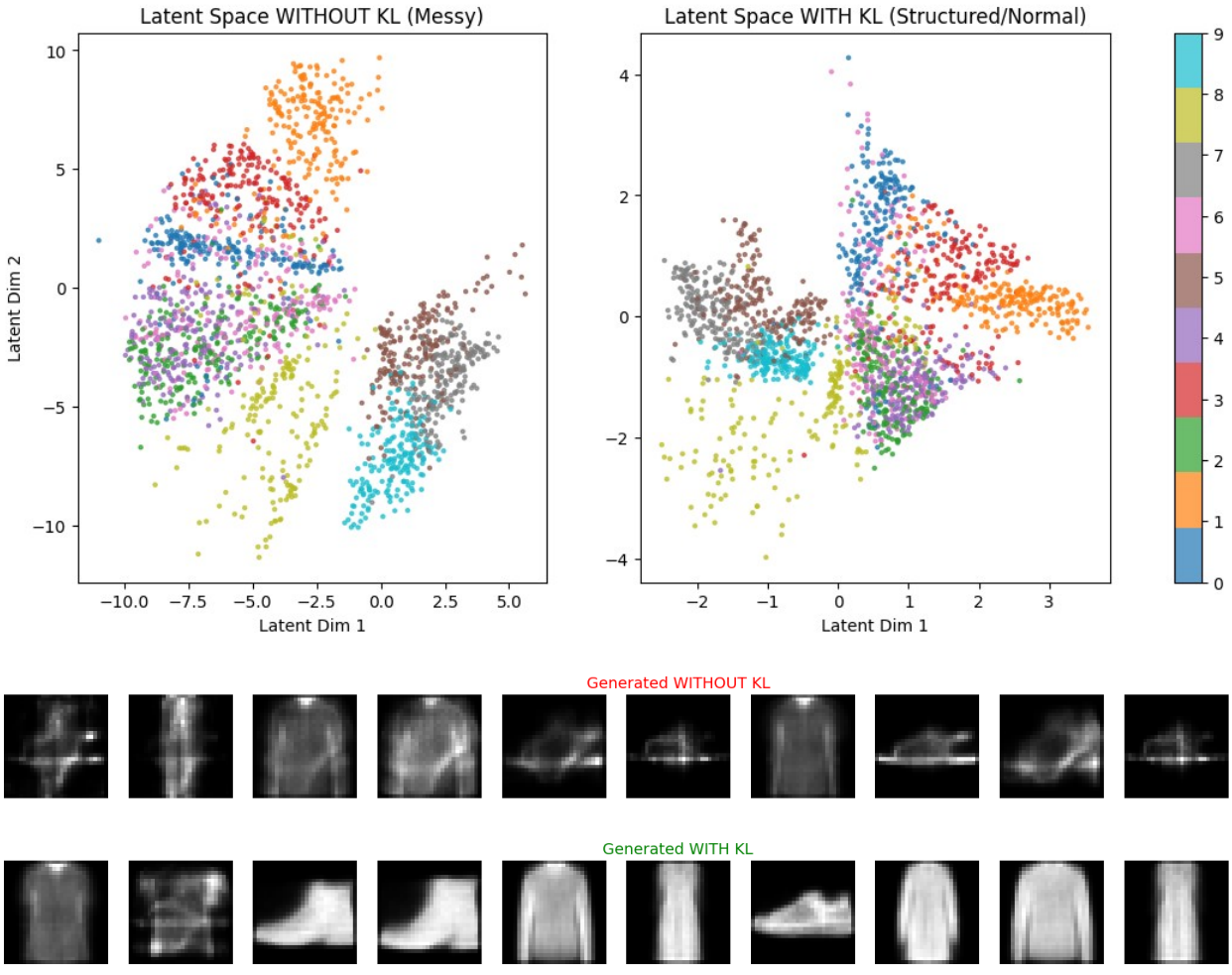
```
Using device: cuda

100%|██████████| 26.4M/26.4M [00:01<00:00, 14.3MB/s]
100%|██████████| 29.5k/29.5k [00:00<00:00, 212kB/s]
100%|██████████| 4.42M/4.42M [00:01<00:00, 3.92MB/s]
100%|██████████| 5.15k/5.15k [00:00<00:00, 14.5MB/s]


--- Starting Training: KLD DISABLED ---
Epoch 1/10 | BCE Loss: 275.97
Epoch 2/10 | BCE Loss: 261.60
Epoch 3/10 | BCE Loss: 258.84
Epoch 4/10 | BCE Loss: 257.37
Epoch 5/10 | BCE Loss: 256.41
Epoch 6/10 | BCE Loss: 255.61
Epoch 7/10 | BCE Loss: 255.09
Epoch 8/10 | BCE Loss: 254.63
Epoch 9/10 | BCE Loss: 254.20
```

```
Epoch 10/10 | BCE Loss: 253.84

--- Starting Training: KLD ENABLED ---
Epoch 1/10 | BCE Loss: 283.28
Epoch 2/10 | BCE Loss: 266.35
Epoch 3/10 | BCE Loss: 263.52
Epoch 4/10 | BCE Loss: 262.01
Epoch 5/10 | BCE Loss: 260.90
Epoch 6/10 | BCE Loss: 259.95
Epoch 7/10 | BCE Loss: 259.21
Epoch 8/10 | BCE Loss: 258.51
Epoch 9/10 | BCE Loss: 257.93
Epoch 10/10 | BCE Loss: 257.35
```




```
Final BCE Loss (Without KL): 253.84 (Focuses only on memorization)
Final BCE Loss (With KL):    257.35 (Slightly higher tradeoff for
valid generation)
```