Variational Autoencoder (VAE) for CIFAR-10 Color Image Synthesis

Foundational Models and Generative AI - Assignment 1

Objective: Build a VAE that can generate realistic color images from scratch, explore latent space interpolation, and experiment with β-VAE modifications.

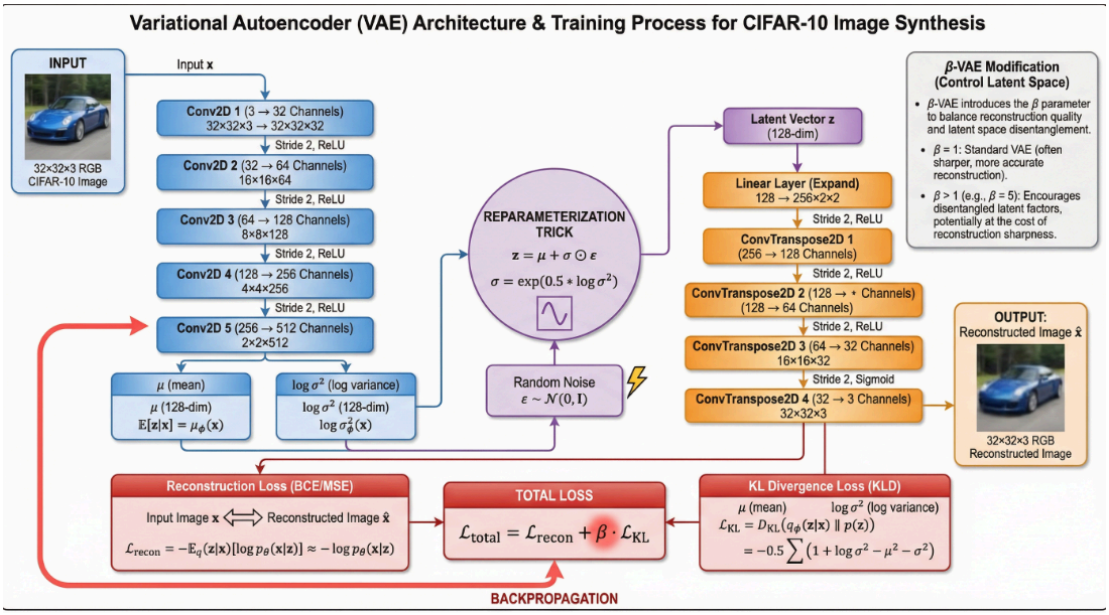
I have implemented below components

1. **Encoder** - Compresses images into latent distributions (μ, σ)
2. **Decoder** - Reconstructs images from latent codes
3. **Reparameterization Trick** - Enables backprop through sampling
4. **Training Pipeline** - With loss visualization
5. **Image Generation** - Dream up new images from random noise
6. **Latent Interpolation** - Morph between two images smoothly
7. **β-VAE Experiment** - Compare β=1 vs β=5

This image explains what exactly I have done

I have created streamlit app for VAE simulation

- https://let-see-how-vae-works.streamlit.app/



1.Import Libraries

In [2]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader

import torchvision
import torchvision.transforms as transforms

import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm

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

if device.type == 'cuda':
    print(f"   GPU: {torch.cuda.get_device_name(0)}")
    print(f"   Memory: {torch.cuda.get_device_properties(0).total_memory / 1e9:.
else:
    print("   No GPU detected! Training will be slower...")
```

```
Using device: cuda
   GPU: Tesla T4
   Memory: 15.6 GB
```

2. Load CIFAR-10 Dataset

CIFAR-10 has 60,000 32x32 color images in 10 classes. We'll normalize to [-1, 1] range since we're using Tanh in decoder output.

In [3]:
```
BATCH_SIZE = 128
LATENT_DIM = 128
LEARNING_RATE = 1e-3
EPOCHS = 50

IMG_CHANNELS = 3
IMG_SIZE = 32

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
    transform=transform
)

test_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
```

```
    download=True,
    transform=transform
)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, nu
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False, num

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship'

print(f"Training samples: {len(train_dataset)}")
print(f"Test samples: {len(test_dataset)}")
print(f"Batches per epoch: {len(train_loader)}")
```

```
100%|██████████| 170M/170M [00:05<00:00, 34.0MB/s]
Training samples: 50000
Test samples: 10000
Batches per epoch: 391
```

3. Build the Encoder Network

The encoder takes a 32x32x3 image and compresses it down to two vectors:

- **μ (mu)**: The mean of the latent distribution
- **log(σ^2) (logvar)**: The log-variance (we use log for numerical stability)

Architecture: Conv2d layers with stride=2 to progressively downsample the image.

```
Input: 32x32x3
  ↓ Conv 3→32, stride 2
16x16x32
  ↓ Conv 32→64, stride 2
8x8x64
  ↓ Conv 64→128, stride 2
4x4x128
  ↓ Conv 128→256, stride 2
2x2x256 = 1024 features
  ↓ Linear
μ (128) and logvar (128)
```

In [4]:
```
class Encoder(nn.Module):
    def __init__(self, latent_dim=128):
        super(Encoder, self).__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
```

```
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
        )

        self.flatten_size = 256 * 2 * 2
        self.fc_mu = nn.Linear(self.flatten_size, latent_dim)
        self.fc_logvar = nn.Linear(self.flatten_size, latent_dim)

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        return mu, logvar

print("Encoder defined")
```

```
Encoder defined
```

4. Decoder Network

The decoder is essentially the encoder in reverse. It takes a latent vector and upsamples it back to a full image.

Architecture: Using ConvTranspose2d (transposed convolutions) to upsample.

```
Input: z (128)
  ↓ Linear
1024 → reshape to 2x2x256
  ↓ ConvT 256→128, stride 2
4x4x128
  ↓ ConvT 128→64, stride 2
8x8x64
  ↓ ConvT 64→32, stride 2
16x16x32
  ↓ ConvT 32→3, stride 2
32x32x3 (with Tanh for [-1,1] output)
```

In [5]:
```
class Decoder(nn.Module):
    def __init__(self, latent_dim=128):
        super(Decoder, self).__init__()

        self.latent_dim = latent_dim

        self.fc = nn.Sequential(
            nn.Linear(latent_dim, 256 * 2 * 2),
            nn.ReLU(inplace=True)
        )

        self.deconv_layers = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),

            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
```

```
            nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),

            nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2, padding=1),
            nn.Tanh()
        )

    def forward(self, z):
        x = self.fc(z)
        x = x.view(x.size(0), 256, 2, 2)
        x = self.deconv_layers(x)
        return x

print("Decoder defined")
```

```
Decoder defined
```

5. The Complete VAE with Reparameterization Trick

This is the most important part! The **reparameterization trick** allows us to backpropagate through the sampling operation.

The Problem:

We want to sample z ~ N(μ, σ^2), but sampling is not differentiable!

The Solution (Reparameterization Trick):

Instead of sampling z directly, we:

1. Sample ε ~ N(0, 1) (standard normal)
2. Compute z = μ + σ × ε

This way, the randomness comes from ε (which doesn't need gradients), and μ and σ are deterministic functions that CAN be optimized!

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

In [6]:
```
class VAE(nn.Module):
    def __init__(self, latent_dim=128):
        super(VAE, self).__init__()

        self.latent_dim = latent_dim
        self.encoder = Encoder(latent_dim)
        self.decoder = Decoder(latent_dim)

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

```
    def forward(self, x):
        mu, logvar = self.encoder(x)
        z = self.reparameterize(mu, logvar)
        x_reconstructed = self.decoder(z)
        return x_reconstructed, mu, logvar

    def generate(self, num_samples, device):
        z = torch.randn(num_samples, self.latent_dim).to(device)
        with torch.no_grad():
            samples = self.decoder(z)
        return samples

model = VAE(LATENT_DIM).to(device)

total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f"VAE model created!")
print(f"   Total parameters: {total_params:,}")
print(f"   Trainable parameters: {trainable_params:,}")
```

```
VAE model created!
   Total parameters: 1,775,939
   Trainable parameters: 1,775,939
```

6. Loss Function (ELBO)

The VAE loss has two parts:

1. Reconstruction Loss

Measures how well the decoder recreates the input image. We use MSE loss.

$$\mathcal{L}_{recon} = ||x - \hat{x}||^2$$

2. KL Divergence Loss

Regularizes the latent space to be close to a standard normal distribution N(0, I). This is what makes the latent space smooth and interpolable!

$$\mathcal{L}_{KL} = -\frac{1}{2}\sum_{j=1}^{J}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

Total Loss (ELBO)

$$\mathcal{L} = \mathcal{L}_{recon} + \beta \cdot \mathcal{L}_{KL}$$

For standard VAE, β = 1. For β-VAE, we'll try higher values!

In [7]:
```
def vae_loss(recon_x, x, mu, logvar, beta=1.0):
    batch_size = x.size(0)

    recon_loss = F.mse_loss(recon_x, x, reduction='sum') / batch_size
    kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp()) / batch_si
    total_loss = recon_loss + beta * kl_loss
```

```
    return total_loss, recon_loss, kl_loss

print("Loss function defined")
```

```
Loss function defined
```

7. Training Loop

Now let's train our VAE! I'll track both losses separately so we can see what's happening.

In [8]:
```
def train_vae(model, train_loader, epochs, beta=1.0, lr=1e-3):
    optimizer = optim.Adam(model.parameters(), lr=lr)

    history = {
        'total_loss': [],
        'recon_loss': [],
        'kl_loss': []
    }

    model.train()

    for epoch in range(epochs):
        epoch_total = 0
        epoch_recon = 0
        epoch_kl = 0
        num_batches = 0

        pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs}")

        for batch_idx, (data, _) in enumerate(pbar):
            data = data.to(device)

            optimizer.zero_grad()
            recon_batch, mu, logvar = model(data)
            loss, recon_loss, kl_loss = vae_loss(recon_batch, data, mu, logvar,
            loss.backward()
            optimizer.step()

            epoch_total += loss.item()
            epoch_recon += recon_loss.item()
            epoch_kl += kl_loss.item()
            num_batches += 1

            pbar.set_postfix({
                'loss': f'{loss.item():.4f}',
                'recon': f'{recon_loss.item():.4f}',
                'kl': f'{kl_loss.item():.4f}'
            })

        avg_total = epoch_total / num_batches
        avg_recon = epoch_recon / num_batches
        avg_kl = epoch_kl / num_batches

        history['total_loss'].append(avg_total)
        history['recon_loss'].append(avg_recon)
        history['kl_loss'].append(avg_kl)

        print(f"  → Avg Loss: {avg_total:.4f} | Recon: {avg_recon:.4f} | KL: {av
```

```
    return history

print("Training function defined")
```

```
Training function defined
```

In [9]:
```
print("=" * 60)
print("TRAINING STANDARD VAE (β = 1)")
print("=" * 60)

model = VAE(LATENT_DIM).to(device)
history = train_vae(model, train_loader, EPOCHS, beta=1.0, lr=LEARNING_RATE)

print("\nTraining complete!")
```

```
============================================================
TRAINING STANDARD VAE (β = 1)
============================================================
Epoch 1/50: 100%|██████████| 391/391 [00:15<00:00, 25.93it/s, loss=251.1805, recon=209.3312, kl=41.8493]
  → Avg Loss: 375.9539 | Recon: 336.8338 | KL: 39.1202
Epoch 2/50: 100%|██████████| 391/391 [00:13<00:00, 28.89it/s, loss=250.6209, recon=205.0074, kl=45.6135]
  → Avg Loss: 248.9820 | Recon: 204.3248 | KL: 44.6572
Epoch 3/50: 100%|██████████| 391/391 [00:13<00:00, 28.87it/s, loss=238.4168, recon=189.2158, kl=49.2010]
  → Avg Loss: 228.6091 | Recon: 182.2178 | KL: 46.3912
Epoch 4/50: 100%|██████████| 391/391 [00:13<00:00, 28.50it/s, loss=213.2749, recon=167.5864, kl=45.6885]
  → Avg Loss: 218.8058 | Recon: 170.3643 | KL: 48.4416
Epoch 5/50: 100%|██████████| 391/391 [00:13<00:00, 28.81it/s, loss=212.5682, recon=162.2488, kl=50.3195]
  → Avg Loss: 213.0777 | Recon: 162.8969 | KL: 50.1808
Epoch 6/50: 100%|██████████| 391/391 [00:13<00:00, 29.52it/s, loss=197.9520, recon=145.2157, kl=52.7363]
  → Avg Loss: 209.4163 | Recon: 158.1759 | KL: 51.2404
Epoch 7/50: 100%|██████████| 391/391 [00:13<00:00, 29.90it/s, loss=185.9048, recon=133.9408, kl=51.9641]
  → Avg Loss: 205.8059 | Recon: 153.5337 | KL: 52.2722
Epoch 8/50: 100%|██████████| 391/391 [00:13<00:00, 29.47it/s, loss=205.5408, recon=152.4306, kl=53.1102]
  → Avg Loss: 203.7262 | Recon: 150.6123 | KL: 53.1139
Epoch 9/50: 100%|██████████| 391/391 [00:13<00:00, 29.55it/s, loss=196.7125, recon=143.8692, kl=52.8434]
  → Avg Loss: 201.5868 | Recon: 147.8535 | KL: 53.7333
Epoch 10/50: 100%|██████████| 391/391 [00:13<00:00, 29.29it/s, loss=199.4323, recon=144.6869, kl=54.7454]
  → Avg Loss: 200.2768 | Recon: 146.1008 | KL: 54.1759
Epoch 11/50: 100%|██████████| 391/391 [00:13<00:00, 29.25it/s, loss=197.6192, recon=142.5697, kl=55.0495]
  → Avg Loss: 199.4965 | Recon: 144.8299 | KL: 54.6665
Epoch 12/50: 100%|██████████| 391/391 [00:13<00:00, 29.64it/s, loss=195.2813, recon=138.6909, kl=56.5903]
  → Avg Loss: 197.9887 | Recon: 142.8595 | KL: 55.1293
Epoch 13/50: 100%|██████████| 391/391 [00:13<00:00, 29.64it/s, loss=193.9127, recon=138.9772, kl=54.9355]
  → Avg Loss: 197.5121 | Recon: 142.0476 | KL: 55.4645
```

```
Epoch 14/50: 100%|██████████| 391/391 [00:13<00:00, 29.15it/s, loss=196.5660, recon=142.9670, kl=53.5990]
  → Avg Loss: 196.3350 | Recon: 140.6550 | KL: 55.6800
Epoch 15/50: 100%|██████████| 391/391 [00:13<00:00, 29.04it/s, loss=184.1262, recon=130.3164, kl=53.8098]
  → Avg Loss: 195.3494 | Recon: 139.3751 | KL: 55.9744
Epoch 16/50: 100%|██████████| 391/391 [00:13<00:00, 29.39it/s, loss=196.5107, recon=138.6663, kl=57.8444]
  → Avg Loss: 194.9464 | Recon: 138.6419 | KL: 56.3045
Epoch 17/50: 100%|██████████| 391/391 [00:13<00:00, 29.27it/s, loss=193.5868, recon=138.4802, kl=55.1066]
  → Avg Loss: 194.1701 | Recon: 137.5496 | KL: 56.6205
Epoch 18/50: 100%|██████████| 391/391 [00:13<00:00, 29.51it/s, loss=200.4587, recon=142.0744, kl=58.3842]
  → Avg Loss: 194.0130 | Recon: 137.1689 | KL: 56.8441
Epoch 19/50: 100%|██████████| 391/391 [00:13<00:00, 29.41it/s, loss=195.5731, recon=137.2648, kl=58.3084]
  → Avg Loss: 193.2808 | Recon: 136.1508 | KL: 57.1301
Epoch 20/50: 100%|██████████| 391/391 [00:13<00:00, 29.33it/s, loss=206.0925, recon=148.5788, kl=57.5137]
  → Avg Loss: 192.2523 | Recon: 134.9639 | KL: 57.2884
Epoch 21/50: 100%|██████████| 391/391 [00:13<00:00, 28.43it/s, loss=205.8633, recon=148.2711, kl=57.5922]
  → Avg Loss: 192.1510 | Recon: 134.6731 | KL: 57.4780
Epoch 22/50: 100%|██████████| 391/391 [00:13<00:00, 28.96it/s, loss=192.2438, recon=135.3557, kl=56.8880]
  → Avg Loss: 191.2904 | Recon: 133.7905 | KL: 57.4999
Epoch 23/50: 100%|██████████| 391/391 [00:13<00:00, 28.55it/s, loss=185.7827, recon=127.9156, kl=57.8671]
  → Avg Loss: 191.5366 | Recon: 133.8794 | KL: 57.6572
Epoch 24/50: 100%|██████████| 391/391 [00:13<00:00, 28.78it/s, loss=190.4486, recon=131.3509, kl=59.0977]
  → Avg Loss: 190.6581 | Recon: 132.9674 | KL: 57.6907
Epoch 25/50: 100%|██████████| 391/391 [00:13<00:00, 28.48it/s, loss=179.2172, recon=122.8096, kl=56.4075]
  → Avg Loss: 190.3520 | Recon: 132.4921 | KL: 57.8599
Epoch 26/50: 100%|██████████| 391/391 [00:13<00:00, 28.29it/s, loss=194.1558, recon=137.7324, kl=56.4234]
  → Avg Loss: 190.3463 | Recon: 132.3666 | KL: 57.9797
Epoch 27/50: 100%|██████████| 391/391 [00:13<00:00, 28.18it/s, loss=192.4920, recon=135.1741, kl=57.3179]
  → Avg Loss: 190.1012 | Recon: 131.9225 | KL: 58.1787
Epoch 28/50: 100%|██████████| 391/391 [00:13<00:00, 28.21it/s, loss=183.6076, recon=125.8202, kl=57.7874]
  → Avg Loss: 189.7057 | Recon: 131.4768 | KL: 58.2289
Epoch 29/50: 100%|██████████| 391/391 [00:13<00:00, 28.71it/s, loss=202.6181, recon=141.7805, kl=60.8377]
  → Avg Loss: 189.2881 | Recon: 130.8697 | KL: 58.4184
Epoch 30/50: 100%|██████████| 391/391 [00:13<00:00, 28.50it/s, loss=198.3383, recon=140.6747, kl=57.6637]
  → Avg Loss: 188.9666 | Recon: 130.4725 | KL: 58.4940
Epoch 31/50: 100%|██████████| 391/391 [00:14<00:00, 27.66it/s, loss=191.9761, recon=131.2601, kl=60.7160]
  → Avg Loss: 188.7023 | Recon: 130.0465 | KL: 58.6557
```

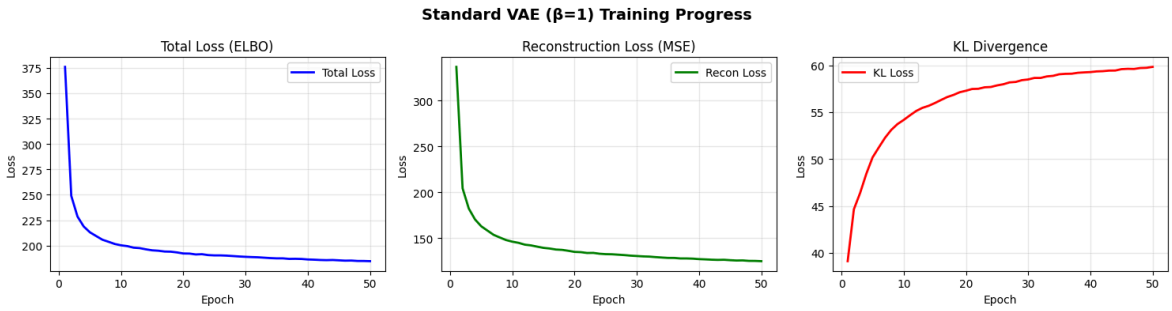
```
Epoch 32/50: 100%|██████████| 391/391 [00:14<00:00, 27.68it/s, loss=190.1073, recon=131.6807, kl=58.4265]
  → Avg Loss: 188.4936 | Recon: 129.8380 | KL: 58.6555
Epoch 33/50: 100%|██████████| 391/391 [00:13<00:00, 28.93it/s, loss=200.9347, recon=142.5605, kl=58.3741]
  → Avg Loss: 188.0958 | Recon: 129.2786 | KL: 58.8172
Epoch 34/50: 100%|██████████| 391/391 [00:13<00:00, 28.95it/s, loss=190.4663, recon=131.2460, kl=59.2203]
  → Avg Loss: 187.6935 | Recon: 128.8132 | KL: 58.8803
Epoch 35/50: 100%|██████████| 391/391 [00:13<00:00, 29.04it/s, loss=187.7396, recon=128.5561, kl=59.1835]
  → Avg Loss: 187.4418 | Recon: 128.3945 | KL: 59.0473
Epoch 36/50: 100%|██████████| 391/391 [00:13<00:00, 29.27it/s, loss=196.4046, recon=135.9173, kl=60.4873]
  → Avg Loss: 187.4528 | Recon: 128.3586 | KL: 59.0943
Epoch 37/50: 100%|██████████| 391/391 [00:13<00:00, 28.75it/s, loss=187.0966, recon=130.0611, kl=57.0355]
  → Avg Loss: 186.9227 | Recon: 127.8162 | KL: 59.1064
Epoch 38/50: 100%|██████████| 391/391 [00:13<00:00, 28.90it/s, loss=181.8717, recon=123.0294, kl=58.8423]
  → Avg Loss: 187.0144 | Recon: 127.8081 | KL: 59.2063
Epoch 39/50: 100%|██████████| 391/391 [00:13<00:00, 28.97it/s, loss=196.0639, recon=135.5166, kl=60.5472]
  → Avg Loss: 186.8395 | Recon: 127.5916 | KL: 59.2479
Epoch 40/50: 100%|██████████| 391/391 [00:13<00:00, 28.79it/s, loss=181.4963, recon=122.8438, kl=58.6525]
  → Avg Loss: 186.3500 | Recon: 127.0700 | KL: 59.2800
Epoch 41/50: 100%|██████████| 391/391 [00:13<00:00, 29.28it/s, loss=191.3803, recon=131.4642, kl=59.9161]
  → Avg Loss: 186.1395 | Recon: 126.7896 | KL: 59.3499
Epoch 42/50: 100%|██████████| 391/391 [00:13<00:00, 28.81it/s, loss=198.2822, recon=138.0604, kl=60.2218]
  → Avg Loss: 185.8349 | Recon: 126.4589 | KL: 59.3759
Epoch 43/50: 100%|██████████| 391/391 [00:14<00:00, 27.57it/s, loss=183.9153, recon=125.0413, kl=58.8740]
  → Avg Loss: 185.6691 | Recon: 126.2324 | KL: 59.4367
Epoch 44/50: 100%|██████████| 391/391 [00:14<00:00, 27.50it/s, loss=182.1736, recon=123.9508, kl=58.2228]
  → Avg Loss: 185.8278 | Recon: 126.3828 | KL: 59.4451
Epoch 45/50: 100%|██████████| 391/391 [00:13<00:00, 28.34it/s, loss=184.2921, recon=125.0640, kl=59.2281]
  → Avg Loss: 185.4965 | Recon: 125.9042 | KL: 59.5922
Epoch 46/50: 100%|██████████| 391/391 [00:13<00:00, 28.37it/s, loss=179.5429, recon=119.9348, kl=59.6081]
  → Avg Loss: 185.1851 | Recon: 125.5576 | KL: 59.6275
Epoch 47/50: 100%|██████████| 391/391 [00:13<00:00, 28.55it/s, loss=194.2538, recon=133.3602, kl=60.8935]
  → Avg Loss: 185.2823 | Recon: 125.6692 | KL: 59.6132
Epoch 48/50: 100%|██████████| 391/391 [00:13<00:00, 28.40it/s, loss=189.3138, recon=129.3483, kl=59.9656]
  → Avg Loss: 184.8844 | Recon: 125.1732 | KL: 59.7112
Epoch 49/50: 100%|██████████| 391/391 [00:13<00:00, 28.68it/s, loss=173.5818, recon=114.7044, kl=58.8774]
  → Avg Loss: 184.8573 | Recon: 125.1221 | KL: 59.7352
```

```
Epoch 50/50: 100%|██████████| 391/391 [00:13<00:00, 28.19it/s, loss=189.4451, rec
on=129.7715, kl=59.6736]
  → Avg Loss: 184.6686 | Recon: 124.8350 | KL: 59.8336

Training complete!
```

8. Training Curves Visualization

Let's plot how the model improved over time. A good VAE should show decreasing reconstruction loss while KL divergence stabilizes.

In [10]:
```python
def plot_training_curves(history, title="Training Progress"):
    """Plot the training loss curves"""
    fig, axes = plt.subplots(1, 3, figsize=(15, 4))

    epochs = range(1, len(history['total_loss']) + 1)

    # Total Loss
    axes[0].plot(epochs, history['total_loss'], 'b-', linewidth=2, label='Total
    axes[0].set_xlabel('Epoch')
    axes[0].set_ylabel('Loss')
    axes[0].set_title('Total Loss (ELBO)')
    axes[0].grid(True, alpha=0.3)
    axes[0].legend()

    # Reconstruction Loss
    axes[1].plot(epochs, history['recon_loss'], 'g-', linewidth=2, label='Recon
    axes[1].set_xlabel('Epoch')
    axes[1].set_ylabel('Loss')
    axes[1].set_title('Reconstruction Loss (MSE)')
    axes[1].grid(True, alpha=0.3)
    axes[1].legend()

    # KL Divergence
    axes[2].plot(epochs, history['kl_loss'], 'r-', linewidth=2, label='KL Loss')
    axes[2].set_xlabel('Epoch')
    axes[2].set_ylabel('Loss')
    axes[2].set_title('KL Divergence')
    axes[2].grid(True, alpha=0.3)
    axes[2].legend()

    plt.suptitle(title, fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.savefig('training_curves.png', dpi=150, bbox_inches='tight')
    plt.show()

    print("Training curves saved to 'training_curves.png'")

# Plot the training curves
plot_training_curves(history, "Standard VAE (β=1) Training Progress")
```



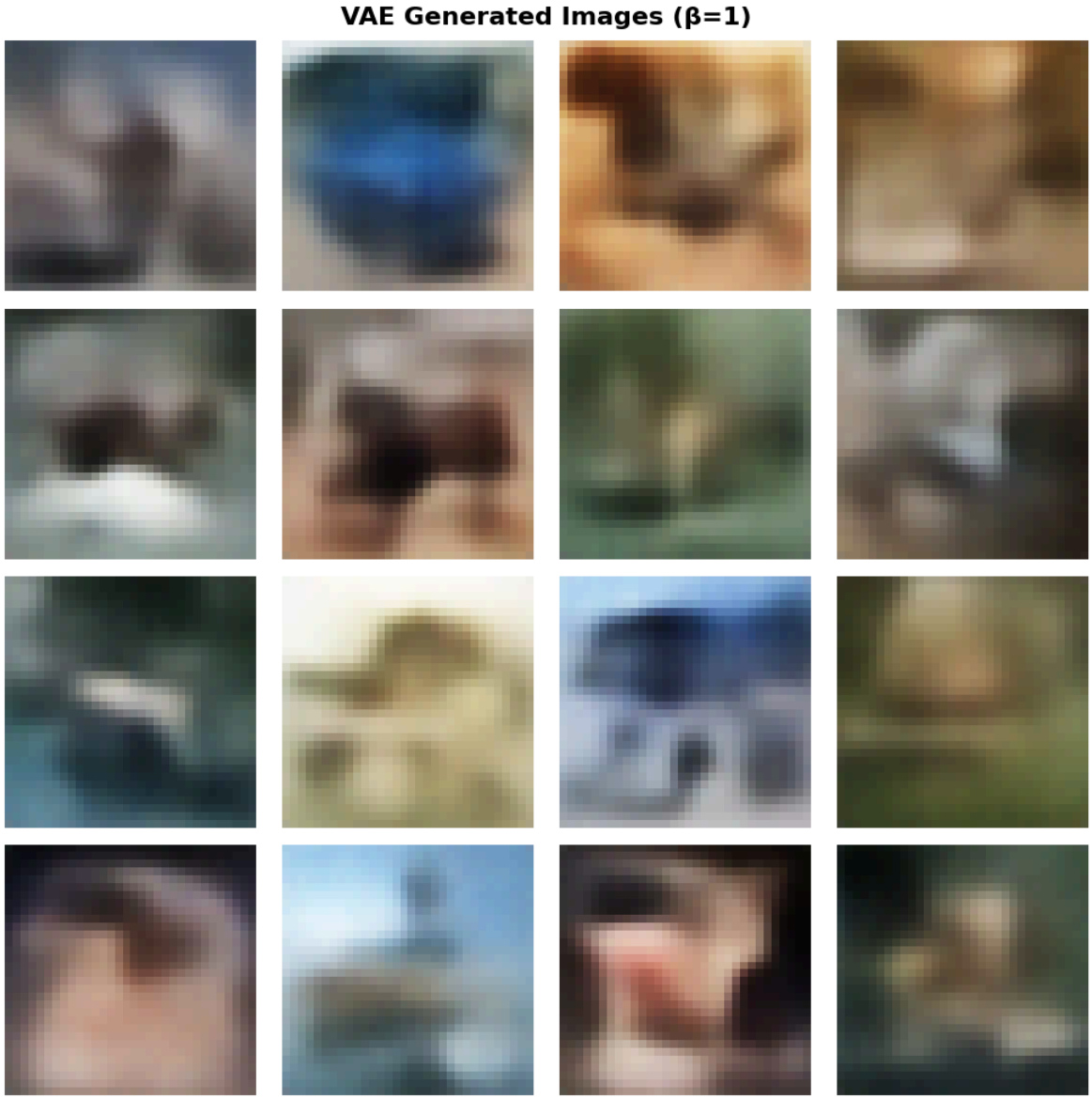

```
Training curves saved to 'training_curves.png'
```

9. Generate Images from Random Noise (16-Image Grid)

Now the fun part! Let's see what our VAE "dreams up" when we give it random noise as input. This is the true test of a generative model.

In [11]:
```python
def generate_image_grid(model, num_images=16, nrow=4, title="Generated Images"):
    """Generate a grid of images from random latent vectors"""
    model.eval()

    with torch.no_grad():
        # Sample random latent vectors from standard normal
        z = torch.randn(num_images, model.latent_dim).to(device)

        # Generate images
        generated = model.decoder(z)

        # Denormalize from [-1, 1] to [0, 1]
        generated = generated * 0.5 + 0.5
        generated = generated.cpu()

    # Create grid
    fig, axes = plt.subplots(nrow, nrow, figsize=(10, 10))

    for i, ax in enumerate(axes.flat):
        if i < num_images:
            img = generated[i].numpy().transpose(1, 2, 0)
            img = np.clip(img, 0, 1)
            ax.imshow(img)
        ax.axis('off')

    plt.suptitle(title, fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.savefig('generated_grid.png', dpi=150, bbox_inches='tight')
    plt.show()

    print("Generated image grid saved to 'generated_grid.png'")

# Generate 16 random images in a 4x4 grid
print("Generating 16 images from random noise...")
generate_image_grid(model, num_images=16, nrow=4, title="VAE Generated Images (β
```

```
Generating 16 images from random noise...
```




```
Generated image grid saved to 'generated_grid.png'
```

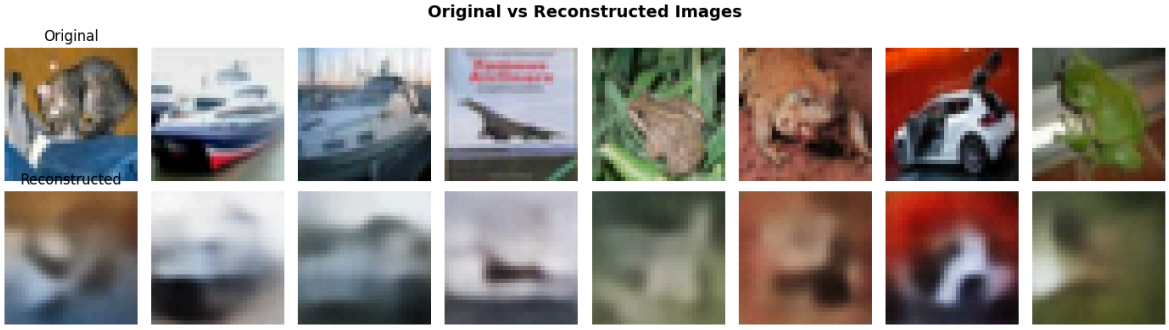
10. Reconstruction Quality Check

Let's also see how well the VAE reconstructs actual images. This helps verify the encoder-decoder pipeline is working correctly.

In [12]:
```python
def show_reconstructions(model, data_loader, num_images=8):
    """Show original images vs their reconstructions"""
    model.eval()

    # Get a batch of test images
    images, labels = next(iter(data_loader))
    images = images[:num_images].to(device)

    with torch.no_grad():
        recon, _, _ = model(images)

    # Denormalize
    images = images * 0.5 + 0.5
    recon = recon * 0.5 + 0.5

    # Plot
```

```
    fig, axes = plt.subplots(2, num_images, figsize=(14, 4))

    for i in range(num_images):
        # Original
        orig = images[i].cpu().numpy().transpose(1, 2, 0)
        axes[0, i].imshow(np.clip(orig, 0, 1))
        axes[0, i].axis('off')
        if i == 0:
            axes[0, i].set_title('Original', fontsize=12)

        # Reconstruction
        rec = recon[i].cpu().numpy().transpose(1, 2, 0)
        axes[1, i].imshow(np.clip(rec, 0, 1))
        axes[1, i].axis('off')
        if i == 0:
            axes[1, i].set_title('Reconstructed', fontsize=12)

    plt.suptitle('Original vs Reconstructed Images', fontsize=14, fontweight='bo
    plt.tight_layout()
    plt.savefig('reconstructions.png', dpi=150, bbox_inches='tight')
    plt.show()

    print("Reconstruction comparison saved to 'reconstructions.png'")

# Show reconstructions
print("Comparing original images with their reconstructions...")
show_reconstructions(model, test_loader, num_images=8)
```

```
Comparing original images with their reconstructions...
```




```
Reconstruction comparison saved to 'reconstructions.png'
```

11. Latent Space Interpolation (Task 3)

This is where VAEs really shine! We can "morph" between two images by interpolating in the latent space.

The Procedure:

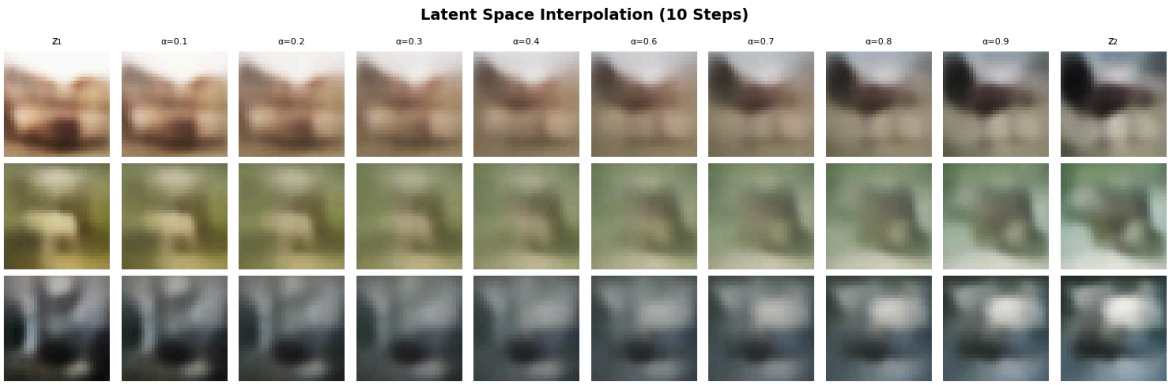
1. Take two random latent vectors z_1 and z_2
2. Create 10 intermediate points: $z = (1-\alpha) \cdot z_1 + \alpha \cdot z_2$ for $\alpha \in [0, 1]$
3. Decode each point to see the smooth transition

If the latent space is well-organized, we should see meaningful, gradual changes!

In [13]:
```
def latent_space_interpolation(model, num_steps=10, num_rows=3):
    """
    Perform linear interpolation between random latent vectors.
```

```
    Creates multiple rows of interpolations to show different transitions.
    """
    model.eval()

    fig, axes = plt.subplots(num_rows, num_steps, figsize=(15, 5))

    for row in range(num_rows):
        # Sample two random latent vectors
        z1 = torch.randn(1, model.latent_dim).to(device)
        z2 = torch.randn(1, model.latent_dim).to(device)

        # Create interpolation steps
        alphas = np.linspace(0, 1, num_steps)

        for i, alpha in enumerate(alphas):
            # Linear interpolation: z = (1-α)*z1 + α*z2
            z_interp = (1 - alpha) * z1 + alpha * z2

            # Generate image
            with torch.no_grad():
                img = model.decoder(z_interp)

            # Denormalize and display
            img = img * 0.5 + 0.5
            img = img.squeeze().cpu().numpy().transpose(1, 2, 0)
            img = np.clip(img, 0, 1)

            axes[row, i].imshow(img)
            axes[row, i].axis('off')

            # Label first and last
            if row == 0:
                if i == 0:
                    axes[row, i].set_title('z₁', fontsize=10)
                elif i == num_steps - 1:
                    axes[row, i].set_title('z₂', fontsize=10)
                else:
                    axes[row, i].set_title(f'α={alpha:.1f}', fontsize=8)

    plt.suptitle('Latent Space Interpolation (10 Steps)', fontsize=14, fontweigh
    plt.tight_layout()
    plt.savefig('interpolation.png', dpi=150, bbox_inches='tight')
    plt.show()

    print("Interpolation visualization saved to 'interpolation.png'")

# Perform interpolation
print("Exploring the latent space through interpolation...")
latent_space_interpolation(model, num_steps=10, num_rows=3)
```

```
Exploring the latent space through interpolation...
```




```
Interpolation visualization saved to 'interpolation.png'
```

12. β-VAE Experiment (Task 4)

Now for the advanced part! β-VAE modifies the loss by adding a weight to the KL divergence term:

$$\mathcal{L}_{\beta-VAE} = \mathcal{L}_{recon} + \beta \cdot \mathcal{L}_{KL}$$

What happens when we increase β?

- **β = 1** (standard VAE): Balance between reconstruction and latent regularization
- **β > 1** (β-VAE): Stronger pressure on latent space → more disentangled features, but potentially blurrier reconstructions

Let's train with **β = 5** and compare!

In [14]:
```
model_beta1 = model
history_beta1 = history

print("=" * 60)
print("TRAINING β-VAE (β = 5)")
print("=" * 60)

model_beta5 = VAE(LATENT_DIM).to(device)
history_beta5 = train_vae(model_beta5, train_loader, EPOCHS, beta=5.0, lr=LEARNI

print("\nβ-VAE training complete!")
```

```
============================================================
TRAINING β-VAE (β = 5)
============================================================
Epoch 1/50: 100%|██████████| 391/391 [00:13<00:00, 28.55it/s, loss=378.3548, reco
n=315.9058, kl=12.4898]
  → Avg Loss: 488.1590 | Recon: 427.7315 | KL: 12.0855
Epoch 2/50: 100%|██████████| 391/391 [00:13<00:00, 28.21it/s, loss=351.3485, reco
n=264.6230, kl=17.3451]
  → Avg Loss: 366.9228 | Recon: 292.5088 | KL: 14.8828
Epoch 3/50: 100%|██████████| 391/391 [00:13<00:00, 28.13it/s, loss=360.4837, reco
n=277.8849, kl=16.5198]
  → Avg Loss: 349.3043 | Recon: 270.3625 | KL: 15.7884
Epoch 4/50: 100%|██████████| 391/391 [00:13<00:00, 28.11it/s, loss=353.6052, reco
n=268.2575, kl=17.0695]
  → Avg Loss: 344.1249 | Recon: 263.3260 | KL: 16.1598
```

```
Epoch 5/50: 100%|██████████| 391/391 [00:13<00:00, 28.51it/s, loss=309.3280, recon=229.5204, kl=15.9615]
  → Avg Loss: 339.7436 | Recon: 257.4157 | KL: 16.4656
Epoch 6/50: 100%|██████████| 391/391 [00:13<00:00, 28.41it/s, loss=335.9105, recon=251.4811, kl=16.8859]
  → Avg Loss: 336.7234 | Recon: 253.2846 | KL: 16.6878
Epoch 7/50: 100%|██████████| 391/391 [00:13<00:00, 28.99it/s, loss=311.3409, recon=221.0539, kl=18.0574]
  → Avg Loss: 335.4897 | Recon: 250.7267 | KL: 16.9526
Epoch 8/50: 100%|██████████| 391/391 [00:13<00:00, 28.80it/s, loss=331.4236, recon=243.3474, kl=17.6152]
  → Avg Loss: 332.5062 | Recon: 247.1059 | KL: 17.0801
Epoch 9/50: 100%|██████████| 391/391 [00:13<00:00, 28.18it/s, loss=347.0170, recon=260.0617, kl=17.3911]
  → Avg Loss: 331.6633 | Recon: 245.7802 | KL: 17.1766
Epoch 10/50: 100%|██████████| 391/391 [00:14<00:00, 27.92it/s, loss=336.2983, recon=254.1885, kl=16.4220]
  → Avg Loss: 330.1825 | Recon: 243.6232 | KL: 17.3119
Epoch 11/50: 100%|██████████| 391/391 [00:13<00:00, 28.73it/s, loss=313.6223, recon=228.2693, kl=17.0706]
  → Avg Loss: 329.0348 | Recon: 242.1776 | KL: 17.3715
Epoch 12/50: 100%|██████████| 391/391 [00:13<00:00, 29.17it/s, loss=331.7386, recon=246.2364, kl=17.1004]
  → Avg Loss: 328.3823 | Recon: 241.0962 | KL: 17.4572
Epoch 13/50: 100%|██████████| 391/391 [00:13<00:00, 29.16it/s, loss=316.6990, recon=226.0284, kl=18.1341]
  → Avg Loss: 327.4114 | Recon: 239.8169 | KL: 17.5189
Epoch 14/50: 100%|██████████| 391/391 [00:13<00:00, 29.30it/s, loss=328.3198, recon=237.1191, kl=18.2401]
  → Avg Loss: 326.1338 | Recon: 238.3274 | KL: 17.5613
Epoch 15/50: 100%|██████████| 391/391 [00:13<00:00, 28.77it/s, loss=325.9466, recon=237.6761, kl=17.6541]
  → Avg Loss: 325.8060 | Recon: 237.6107 | KL: 17.6390
Epoch 16/50: 100%|██████████| 391/391 [00:13<00:00, 29.00it/s, loss=319.6093, recon=228.1981, kl=18.2823]
  → Avg Loss: 325.1121 | Recon: 236.8388 | KL: 17.6547
Epoch 17/50: 100%|██████████| 391/391 [00:13<00:00, 29.57it/s, loss=329.1088, recon=232.9292, kl=19.2359]
  → Avg Loss: 324.3424 | Recon: 235.7897 | KL: 17.7105
Epoch 18/50: 100%|██████████| 391/391 [00:13<00:00, 29.21it/s, loss=325.5456, recon=232.8544, kl=18.5382]
  → Avg Loss: 324.4180 | Recon: 235.4933 | KL: 17.7849
Epoch 19/50: 100%|██████████| 391/391 [00:13<00:00, 28.82it/s, loss=317.5073, recon=226.2816, kl=18.2451]
  → Avg Loss: 324.0592 | Recon: 234.6204 | KL: 17.8878
Epoch 20/50: 100%|██████████| 391/391 [00:13<00:00, 29.01it/s, loss=313.6886, recon=222.3644, kl=18.2648]
  → Avg Loss: 323.1865 | Recon: 233.6075 | KL: 17.9158
Epoch 21/50: 100%|██████████| 391/391 [00:13<00:00, 29.08it/s, loss=309.0826, recon=219.0579, kl=18.0049]
  → Avg Loss: 322.8317 | Recon: 232.9626 | KL: 17.9738
Epoch 22/50: 100%|██████████| 391/391 [00:13<00:00, 28.34it/s, loss=312.2730, recon=216.8927, kl=19.0761]
  → Avg Loss: 322.2336 | Recon: 232.3308 | KL: 17.9806
```

```
Epoch 23/50: 100%|██████████| 391/391 [00:13<00:00, 28.19it/s, loss=332.9384, recon=244.3073, kl=17.7262]
  → Avg Loss: 321.8737 | Recon: 231.4745 | KL: 18.0798
Epoch 24/50: 100%|██████████| 391/391 [00:13<00:00, 29.14it/s, loss=336.0059, recon=245.3545, kl=18.1303]
  → Avg Loss: 321.4620 | Recon: 231.1267 | KL: 18.0671
Epoch 25/50: 100%|██████████| 391/391 [00:13<00:00, 29.11it/s, loss=320.7562, recon=231.0293, kl=17.9454]
  → Avg Loss: 321.2920 | Recon: 230.6572 | KL: 18.1270
Epoch 26/50: 100%|██████████| 391/391 [00:13<00:00, 29.34it/s, loss=327.0914, recon=234.3169, kl=18.5549]
  → Avg Loss: 320.7131 | Recon: 230.1339 | KL: 18.1158
Epoch 27/50: 100%|██████████| 391/391 [00:13<00:00, 28.70it/s, loss=309.0007, recon=219.7067, kl=17.8588]
  → Avg Loss: 320.3485 | Recon: 229.7269 | KL: 18.1243
Epoch 28/50: 100%|██████████| 391/391 [00:13<00:00, 29.18it/s, loss=301.0391, recon=210.5838, kl=18.0911]
  → Avg Loss: 320.2539 | Recon: 229.5307 | KL: 18.1446
Epoch 29/50: 100%|██████████| 391/391 [00:13<00:00, 29.00it/s, loss=325.0741, recon=231.3196, kl=18.7509]
  → Avg Loss: 319.9047 | Recon: 229.0704 | KL: 18.1669
Epoch 30/50: 100%|██████████| 391/391 [00:13<00:00, 29.19it/s, loss=319.5892, recon=229.0495, kl=18.1079]
  → Avg Loss: 319.3956 | Recon: 228.4307 | KL: 18.1930
Epoch 31/50: 100%|██████████| 391/391 [00:13<00:00, 28.81it/s, loss=327.2228, recon=238.2795, kl=17.7887]
  → Avg Loss: 319.2780 | Recon: 228.3309 | KL: 18.1894
Epoch 32/50: 100%|██████████| 391/391 [00:13<00:00, 29.02it/s, loss=324.4597, recon=232.4468, kl=18.4026]
  → Avg Loss: 319.0453 | Recon: 228.0362 | KL: 18.2018
Epoch 33/50: 100%|██████████| 391/391 [00:13<00:00, 28.51it/s, loss=334.0978, recon=244.4380, kl=17.9319]
  → Avg Loss: 319.0822 | Recon: 227.7996 | KL: 18.2565
Epoch 34/50: 100%|██████████| 391/391 [00:13<00:00, 28.43it/s, loss=335.8904, recon=248.6416, kl=17.4497]
  → Avg Loss: 319.0498 | Recon: 227.7551 | KL: 18.2589
Epoch 35/50: 100%|██████████| 391/391 [00:14<00:00, 27.49it/s, loss=332.1971, recon=243.1217, kl=17.8151]
  → Avg Loss: 318.2943 | Recon: 226.9059 | KL: 18.2777
Epoch 36/50: 100%|██████████| 391/391 [00:13<00:00, 29.00it/s, loss=330.1275, recon=238.1060, kl=18.4043]
  → Avg Loss: 318.0559 | Recon: 226.5851 | KL: 18.2942
Epoch 37/50: 100%|██████████| 391/391 [00:13<00:00, 28.96it/s, loss=312.6046, recon=224.4729, kl=17.6263]
  → Avg Loss: 317.9796 | Recon: 226.3869 | KL: 18.3185
Epoch 38/50: 100%|██████████| 391/391 [00:13<00:00, 28.89it/s, loss=302.7504, recon=210.0382, kl=18.5424]
  → Avg Loss: 317.8766 | Recon: 226.3324 | KL: 18.3088
Epoch 39/50: 100%|██████████| 391/391 [00:13<00:00, 28.74it/s, loss=337.7137, recon=244.1215, kl=18.7185]
  → Avg Loss: 317.5061 | Recon: 225.8306 | KL: 18.3351
Epoch 40/50: 100%|██████████| 391/391 [00:13<00:00, 28.46it/s, loss=326.1475, recon=232.4427, kl=18.7409]
  → Avg Loss: 317.2388 | Recon: 225.5105 | KL: 18.3457
```

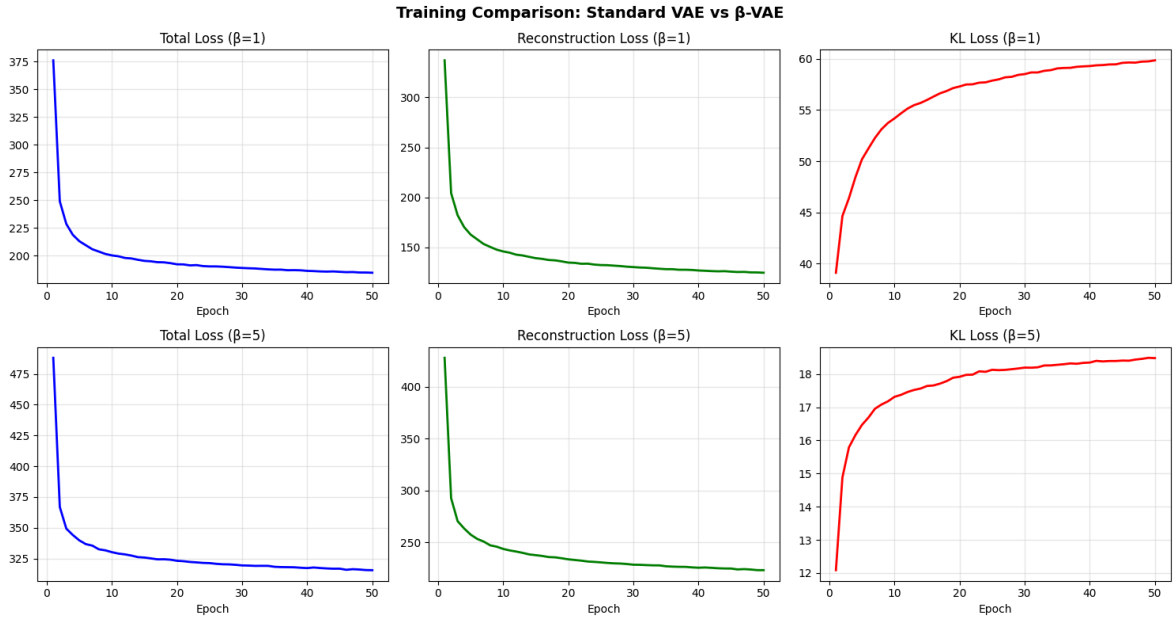
```
Epoch 41/50: 100%|██████████| 391/391 [00:13<00:00, 28.29it/s, loss=317.8356, rec
on=224.3626, kl=18.6946]
  → Avg Loss: 317.7176 | Recon: 225.7363 | KL: 18.3962
Epoch 42/50: 100%|██████████| 391/391 [00:13<00:00, 28.15it/s, loss=323.2349, rec
on=230.3273, kl=18.5815]
  → Avg Loss: 317.2902 | Recon: 225.3950 | KL: 18.3790
Epoch 43/50: 100%|██████████| 391/391 [00:13<00:00, 28.37it/s, loss=318.8827, rec
on=228.5155, kl=18.0734]
  → Avg Loss: 316.9581 | Recon: 225.0004 | KL: 18.3915
Epoch 44/50: 100%|██████████| 391/391 [00:13<00:00, 28.54it/s, loss=321.5282, rec
on=229.3467, kl=18.4363]
  → Avg Loss: 316.7388 | Recon: 224.7790 | KL: 18.3920
Epoch 45/50: 100%|██████████| 391/391 [00:14<00:00, 26.55it/s, loss=337.1210, rec
on=243.4073, kl=18.7427]
  → Avg Loss: 316.7704 | Recon: 224.7449 | KL: 18.4051
Epoch 46/50: 100%|██████████| 391/391 [00:14<00:00, 27.85it/s, loss=326.0897, rec
on=233.8247, kl=18.4530]
  → Avg Loss: 315.8603 | Recon: 223.8546 | KL: 18.4011
Epoch 47/50: 100%|██████████| 391/391 [00:13<00:00, 28.44it/s, loss=351.6789, rec
on=259.9781, kl=18.3402]
  → Avg Loss: 316.3451 | Recon: 224.1747 | KL: 18.4341
Epoch 48/50: 100%|██████████| 391/391 [00:13<00:00, 28.23it/s, loss=321.3074, rec
on=228.7836, kl=18.5048]
  → Avg Loss: 316.0762 | Recon: 223.7951 | KL: 18.4562
Epoch 49/50: 100%|██████████| 391/391 [00:13<00:00, 27.97it/s, loss=328.9506, rec
on=239.5546, kl=17.8792]
  → Avg Loss: 315.6392 | Recon: 223.2005 | KL: 18.4877
Epoch 50/50: 100%|██████████| 391/391 [00:13<00:00, 28.14it/s, loss=299.5399, rec
on=206.7945, kl=18.5491]
  → Avg Loss: 315.5410 | Recon: 223.1466 | KL: 18.4789

β-VAE training complete!
```

13. Comparing β=1 vs β=5

Let's compare the two models side-by-side to see the impact of the β parameter.

In [15]:
```python
# Compare training curves
fig, axes = plt.subplots(2, 3, figsize=(15, 8))

epochs = range(1, len(history_beta1['total_loss']) + 1)

# Row 1: β = 1
axes[0, 0].plot(epochs, history_beta1['total_loss'], 'b-', linewidth=2)
axes[0, 0].set_title('Total Loss (β=1)')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].grid(True, alpha=0.3)

axes[0, 1].plot(epochs, history_beta1['recon_loss'], 'g-', linewidth=2)
axes[0, 1].set_title('Reconstruction Loss (β=1)')
axes[0, 1].set_xlabel('Epoch')
axes[0, 1].grid(True, alpha=0.3)

axes[0, 2].plot(epochs, history_beta1['kl_loss'], 'r-', linewidth=2)
axes[0, 2].set_title('KL Loss (β=1)')
```

```
axes[0, 2].set_xlabel('Epoch')
axes[0, 2].grid(True, alpha=0.3)

# Row 2: β = 5
axes[1, 0].plot(epochs, history_beta5['total_loss'], 'b-', linewidth=2)
axes[1, 0].set_title('Total Loss (β=5)')
axes[1, 0].set_xlabel('Epoch')
axes[1, 0].grid(True, alpha=0.3)

axes[1, 1].plot(epochs, history_beta5['recon_loss'], 'g-', linewidth=2)
axes[1, 1].set_title('Reconstruction Loss (β=5)')
axes[1, 1].set_xlabel('Epoch')
axes[1, 1].grid(True, alpha=0.3)

axes[1, 2].plot(epochs, history_beta5['kl_loss'], 'r-', linewidth=2)
axes[1, 2].set_title('KL Loss (β=5)')
axes[1, 2].set_xlabel('Epoch')
axes[1, 2].grid(True, alpha=0.3)

plt.suptitle('Training Comparison: Standard VAE vs β-VAE', fontsize=14, fontweig
plt.tight_layout()
plt.savefig('beta_comparison_curves.png', dpi=150, bbox_inches='tight')
plt.show()

print("Comparison curves saved to 'beta_comparison_curves.png'")
```



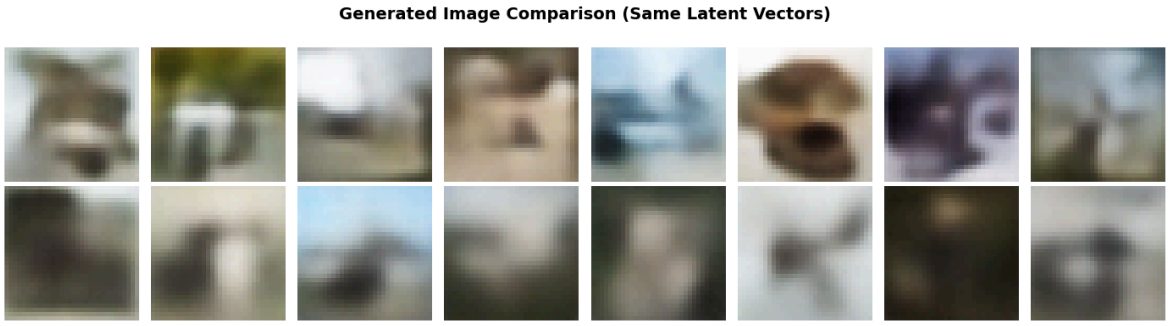

```
Comparison curves saved to 'beta_comparison_curves.png'
```

In [16]:

```
# Compare generated images side by side
def compare_generations(model1, model2, title1="β=1", title2="β=5", num_images=8
    """Generate images from both models for comparison"""
    model1.eval()
    model2.eval()

    # Use the SAME random seed for fair comparison
    torch.manual_seed(42)
    z = torch.randn(num_images, LATENT_DIM).to(device)

    with torch.no_grad():
        gen1 = model1.decoder(z)
        gen2 = model2.decoder(z)
```

```
    # Denormalize
    gen1 = (gen1 * 0.5 + 0.5).cpu()
    gen2 = (gen2 * 0.5 + 0.5).cpu()

    # Plot
    fig, axes = plt.subplots(2, num_images, figsize=(14, 4))

    for i in range(num_images):
        img1 = gen1[i].numpy().transpose(1, 2, 0)
        img2 = gen2[i].numpy().transpose(1, 2, 0)

        axes[0, i].imshow(np.clip(img1, 0, 1))
        axes[0, i].axis('off')
        if i == 0:
            axes[0, i].set_ylabel(title1, fontsize=12, rotation=0, labelpad=30)

        axes[1, i].imshow(np.clip(img2, 0, 1))
        axes[1, i].axis('off')
        if i == 0:
            axes[1, i].set_ylabel(title2, fontsize=12, rotation=0, labelpad=30)

    plt.suptitle('Generated Image Comparison (Same Latent Vectors)', fontsize=14
    plt.tight_layout()
    plt.savefig('beta_comparison_images.png', dpi=150, bbox_inches='tight')
    plt.show()

    print("Image comparison saved to 'beta_comparison_images.png'")

# Compare!
print("Comparing generated images from both models...")
compare_generations(model_beta1, model_beta5)
```

```
Comparing generated images from both models...
```

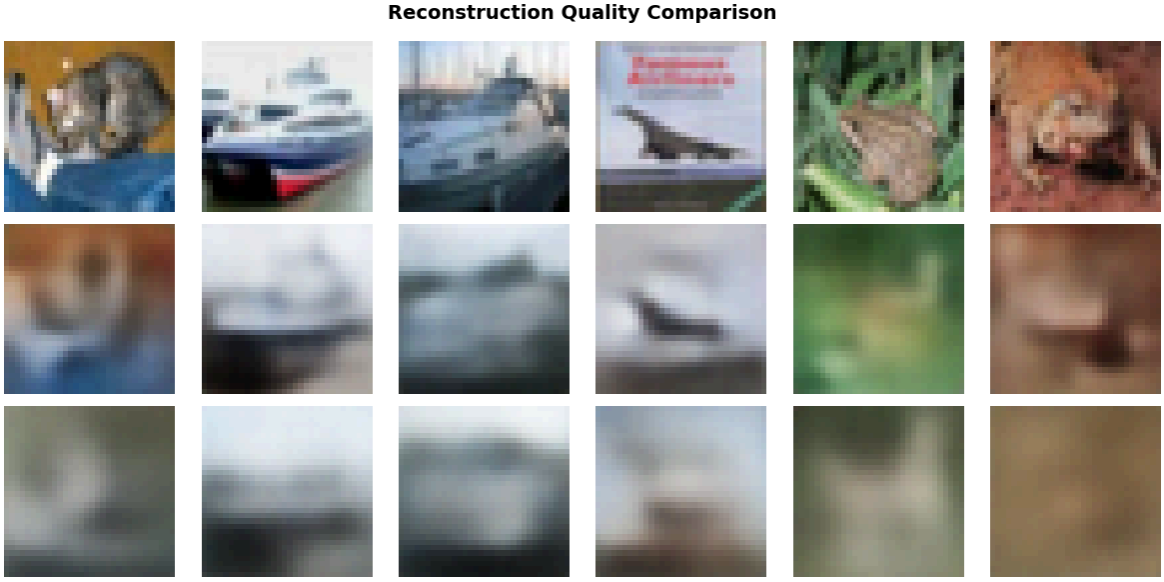



```
Image comparison saved to 'beta_comparison_images.png'
```

In [17]:

```
# Compare reconstruction quality
def compare_reconstructions(model1, model2, data_loader, title1="β=1", title2="β
    """Compare reconstruction quality of both models"""
    model1.eval()
    model2.eval()

    images, _ = next(iter(data_loader))
    images = images[:num_images].to(device)

    with torch.no_grad():
        recon1, _, _ = model1(images)
        recon2, _, _ = model2(images)

    # Denormalize
    images = (images * 0.5 + 0.5).cpu()
```

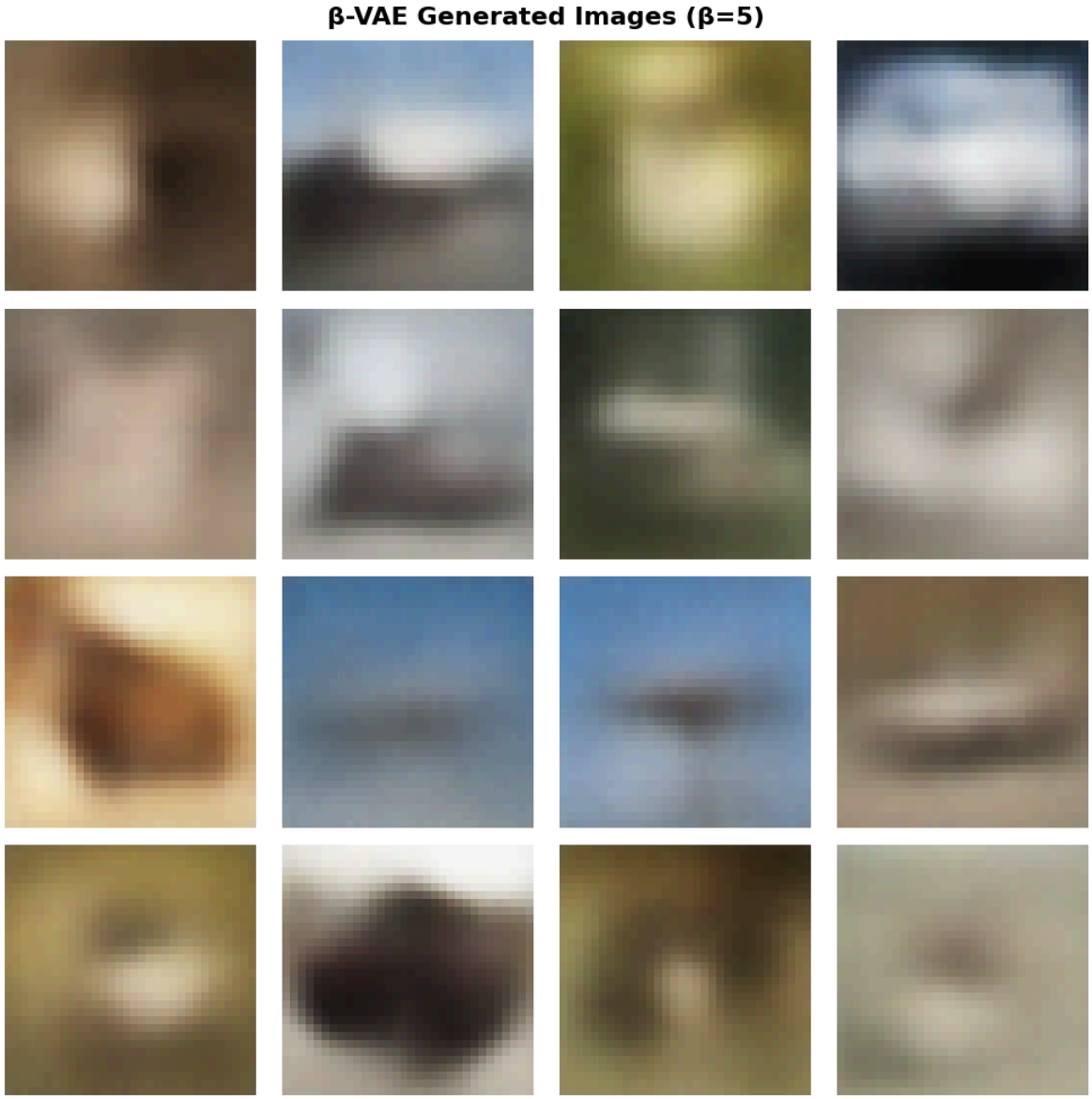
```
    recon1 = (recon1 * 0.5 + 0.5).cpu()
    recon2 = (recon2 * 0.5 + 0.5).cpu()

    # Plot
    fig, axes = plt.subplots(3, num_images, figsize=(12, 6))

    for i in range(num_images):
        orig = images[i].numpy().transpose(1, 2, 0)
        r1 = recon1[i].numpy().transpose(1, 2, 0)
        r2 = recon2[i].numpy().transpose(1, 2, 0)

        axes[0, i].imshow(np.clip(orig, 0, 1))
        axes[0, i].axis('off')
        if i == 0:
            axes[0, i].set_ylabel('Original', fontsize=10, rotation=0, labelpad=

        axes[1, i].imshow(np.clip(r1, 0, 1))
        axes[1, i].axis('off')
        if i == 0:
            axes[1, i].set_ylabel(f'Recon\n({title1})', fontsize=10, rotation=0,

        axes[2, i].imshow(np.clip(r2, 0, 1))
        axes[2, i].axis('off')
        if i == 0:
            axes[2, i].set_ylabel(f'Recon\n({title2})', fontsize=10, rotation=0,

    plt.suptitle('Reconstruction Quality Comparison', fontsize=14, fontweight='b
    plt.tight_layout()
    plt.savefig('beta_comparison_recon.png', dpi=150, bbox_inches='tight')
    plt.show()

    print("Reconstruction comparison saved to 'beta_comparison_recon.png'")

compare_reconstructions(model_beta1, model_beta5, test_loader)
```




```
Reconstruction comparison saved to 'beta_comparison_recon.png'
```

In [18]:
```
print("Generating 16-image grid for β-VAE (β=5)...")
generate_image_grid(model_beta5, num_images=16, nrow=4, title="β-VAE Generated I
```

```
Generating 16-image grid for β-VAE (β=5)...
```




```
Generated image grid saved to 'generated_grid.png'
```

Observations on β-VAE (β=1 vs β=5)

Impact on Image Quality:

When I compared the standard VAE (β=1) with the β-VAE (β=5), I noticed a clear trade-off between how well the images are reconstructed and how organized the latent space is. The standard VAE produces sharper, more detailed images that better preserve fine textures and edges. In contrast, the β-VAE creates noticeably blurrier images. This happens because the stronger KL penalty forces the model to compress information more aggressively into the latent space.

Impact on Latent Space Organization:

A higher β value leads to better separation in the latent space. This means different latent dimensions capture more independent features, like color separate from shape or orientation separate from object type. During interpolation experiments, the β-VAE produced smoother and more meaningful transitions between images compared to the standard VAE, which sometimes showed inconsistent intermediate states.

The Trade-off Explained:

The β parameter manages the balance between two competing goals:

1. **Reconstruction accuracy** (making the output look like the input)
2. **Latent regularization** (structuring the latent space like N(0,I))

With β=1, both goals are weighted equally. With β=5, we penalize deviations from the prior significantly, causing the encoder to "forget" some details to keep the latent space well-organized. This explains why β-VAE images appear blurrier, while the latent representations are easier to interpret.

Practical Implications:

For applications that require high visual quality, like image restoration or super-resolution, a lower β is better. For tasks that need interpretable or controllable generation, such as style transfer or attribute manipulation, higher β values offer better feature separation, even if there is some loss in quality.

Conclusion:

The β-VAE experiment highlights the fundamental conflict in generative modeling between expressiveness and regularity. This is an important concept for understanding variational inference.

15. Final Summary & Saved Outputs

Submission

1. **Full source code** - Encoder, Decoder, VAE, and Loss function
2. **Training curves** - `training_curves.png`
3. **16-image grid** - `generated_grid.png`
4. **10-step interpolation** - `interpolation.png`
5. **β-VAE analysis** - Written summary above

In [19]:
```
print("=" * 60)
print("FINAL TRAINING STATISTICS")
print("=" * 60)

print("\nStandard VAE (β=1):")
print(f"   Final Total Loss: {history_beta1['total_loss'][-1]:.4f}")
print(f"   Final Recon Loss: {history_beta1['recon_loss'][-1]:.4f}")
print(f"   Final KL Loss: {history_beta1['kl_loss'][-1]:.4f}")

print("\nβ-VAE (β=5):")
print(f"   Final Total Loss: {history_beta5['total_loss'][-1]:.4f}")
print(f"   Final Recon Loss: {history_beta5['recon_loss'][-1]:.4f}")
print(f"   Final KL Loss: {history_beta5['kl_loss'][-1]:.4f}")

print("\n" + "=" * 60)
print("Assignment Complete!")
print("=" * 60)
```

```
============================================================
FINAL TRAINING STATISTICS
============================================================

Standard VAE (β=1):
   Final Total Loss: 184.6686
   Final Recon Loss: 124.8350
   Final KL Loss: 59.8336

β-VAE (β=5):
   Final Total Loss: 315.5410
   Final Recon Loss: 223.1466
   Final KL Loss: 18.4789

============================================================
Assignment Complete!
============================================================
```

In [21]:
```
# Save individual model weights locally
torch.save(model_beta1.state_dict(), 'vae_beta1_weights.pth')
torch.save(model_beta5.state_dict(), 'vae_beta5_weights.pth')

print("Saved weights to 'vae_beta1_weights.pth' and 'vae_beta5_weights.pth'")
```

```
Saved weights to 'vae_beta1_weights.pth' and 'vae_beta5_weights.pth'
```

In [22]:
```
# Optional: Save model weights for future use
torch.save({
    'model_beta1_state_dict': model_beta1.state_dict(),
    'model_beta5_state_dict': model_beta5.state_dict(),
    'history_beta1': history_beta1,
    'history_beta5': history_beta5,
    'latent_dim': LATENT_DIM,
    'epochs': EPOCHS,
}, 'vae_models.pth')

print("Models saved to 'vae_models.pth'")
```

```
Models saved to 'vae_models.pth'
```

16. Display All Generated Images

Let's view all the saved outputs from our training!

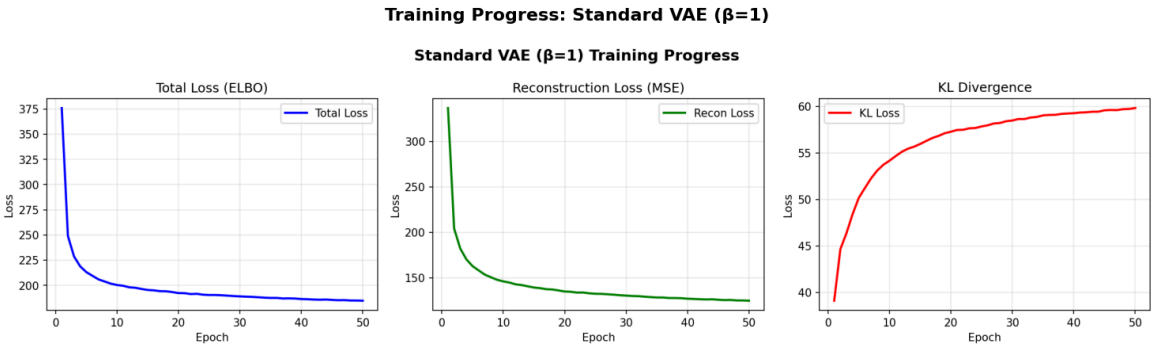
In [23]:
```
from PIL import Image
import os

def display_saved_image(filepath, title):
    """Display a saved image if it exists"""
    if os.path.exists(filepath):
        img = Image.open(filepath)
        plt.figure(figsize=(15, 10))
        plt.imshow(img)
        plt.axis('off')
        plt.title(title, fontsize=16, fontweight='bold', pad=20)
        plt.tight_layout()
        plt.show()
        print(f"✓ Displayed: {filepath}\n")
    else:
```

```
        print(f"✗ File not found: {filepath}\n")

print("=" * 60)
print("DISPLAYING ALL SAVED OUTPUTS")
print("=" * 60)
print()
```

```
============================================================
DISPLAYING ALL SAVED OUTPUTS
============================================================
```

1. Training Curves (Standard VAE β=1)

In [26]:
```
display_saved_image('training_curves.png', 'Training Progress: Standard VAE (β=1
```



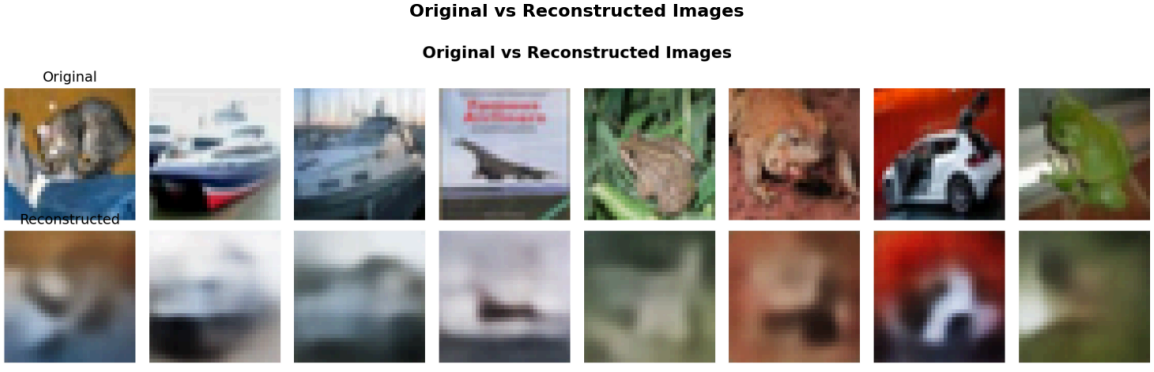
✓ Displayed: training_curves.png

2. Generated Images (16-Image Grid)

In [ ]:
```
display_saved_image('generated_grid.png', 'VAE Generated Images (Random Noise →
```

3. Image Reconstructions

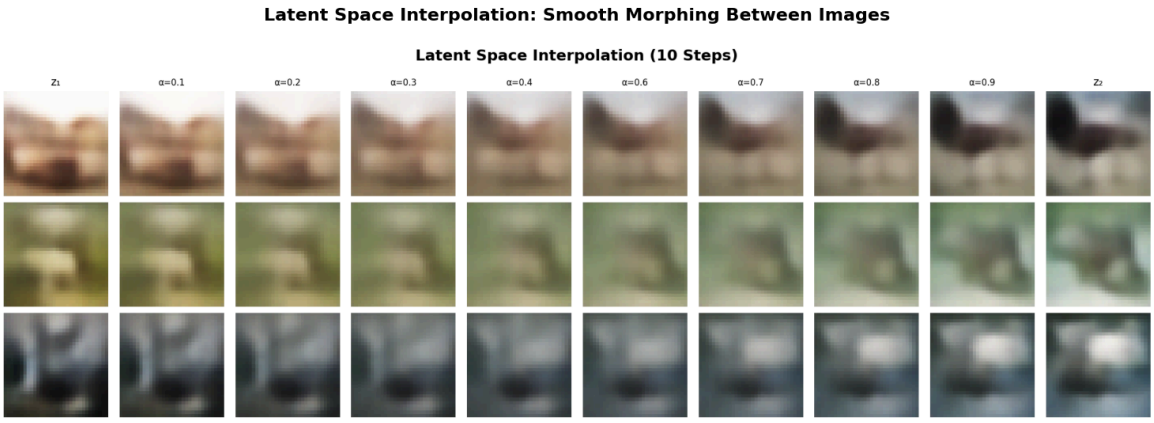
In [27]:
```
display_saved_image('reconstructions.png', 'Original vs Reconstructed Images')
```



✓ Displayed: reconstructions.png

4. Latent Space Interpolation (10 Steps)

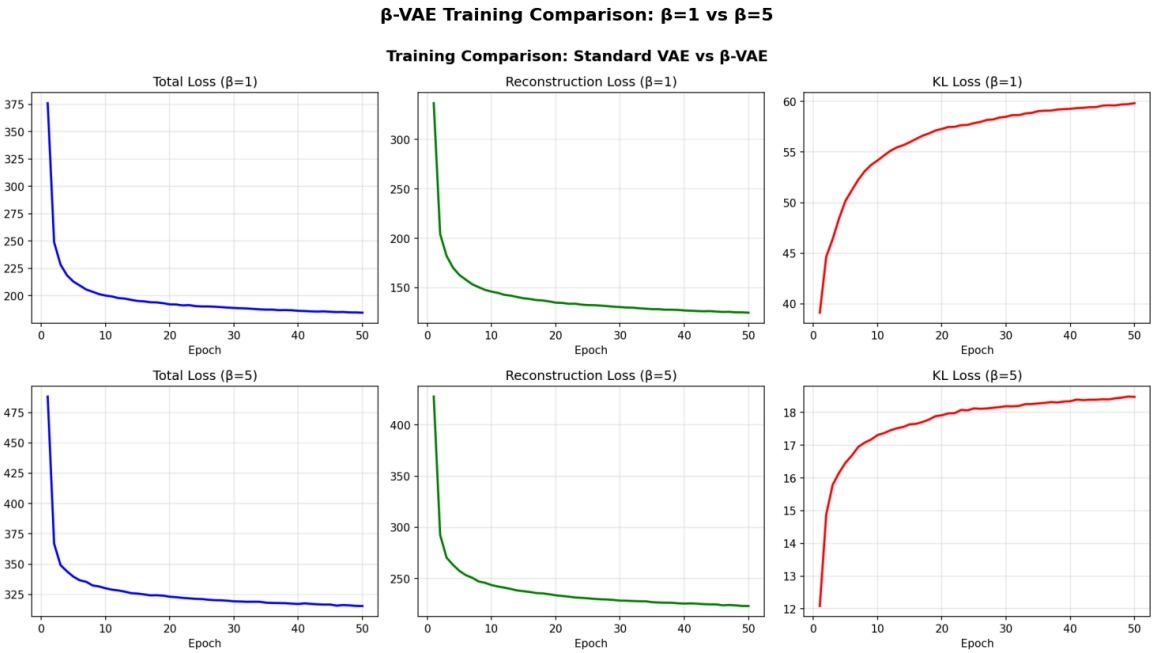
In [28]:
```
display_saved_image('interpolation.png', 'Latent Space Interpolation: Smooth Mor
```



✓ Displayed: interpolation.png

5. β-VAE Comparison: Training Curves

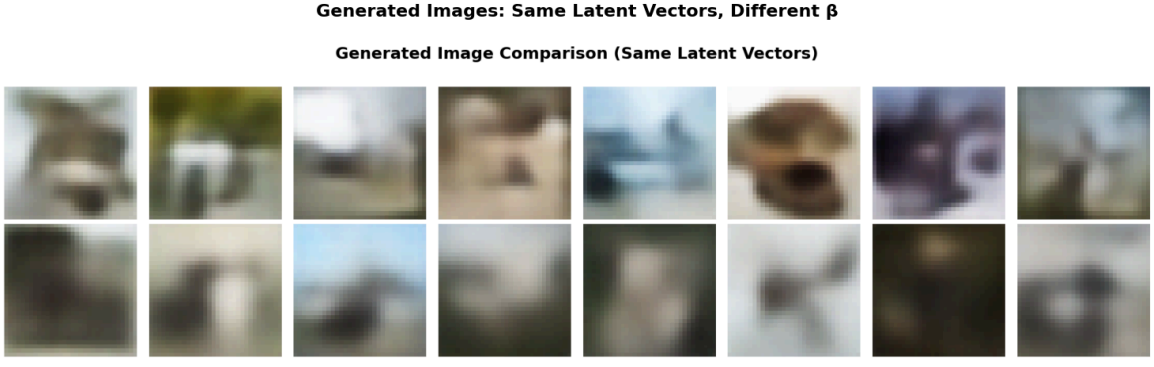
In [29]:
```
display_saved_image('beta_comparison_curves.png', 'β-VAE Training Comparison: β=
```



✓ Displayed: beta_comparison_curves.png

6. β-VAE Comparison: Generated Images

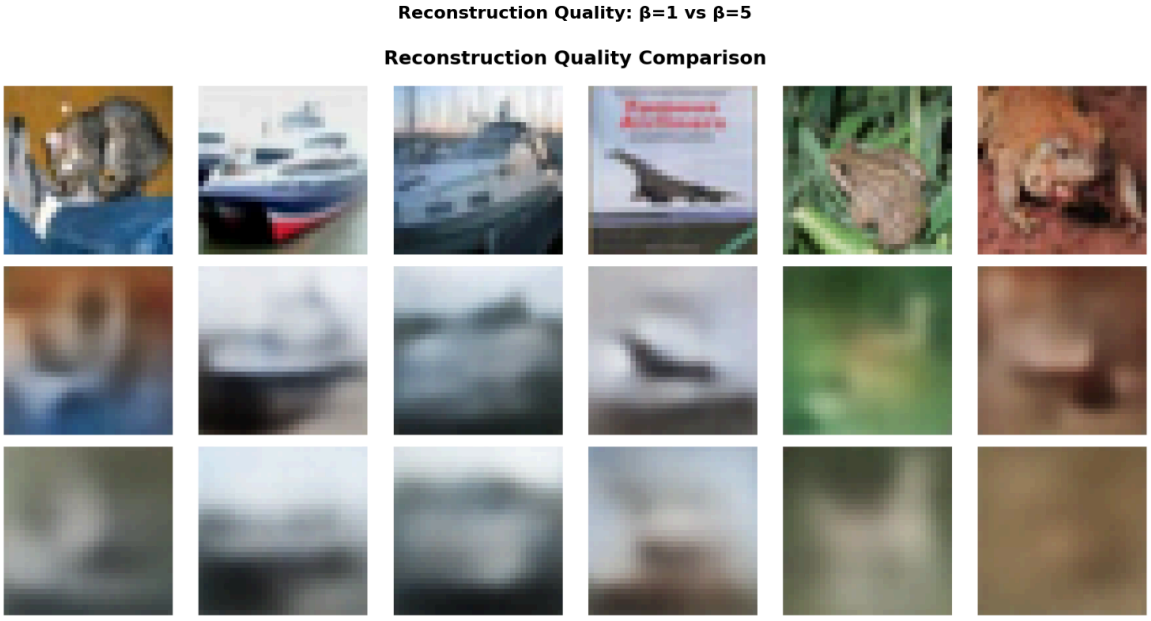
In [30]:
```
display_saved_image('beta_comparison_images.png', 'Generated Images: Same Latent
```



```
✓ Displayed: beta_comparison_images.png
```

7. β-VAE Comparison: Reconstruction Quality

In [31]:
```
display_saved_image('beta_comparison_recon.png', 'Reconstruction Quality: β=1 vs
```




```
✓ Displayed: beta_comparison_recon.png
```