```
!pip install torch torchvision matplotlib
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packages (0.21.0+cu124)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch) (4.13.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from torchvision) (2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from torchvision) (11.2.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch) (3.0.2)
```

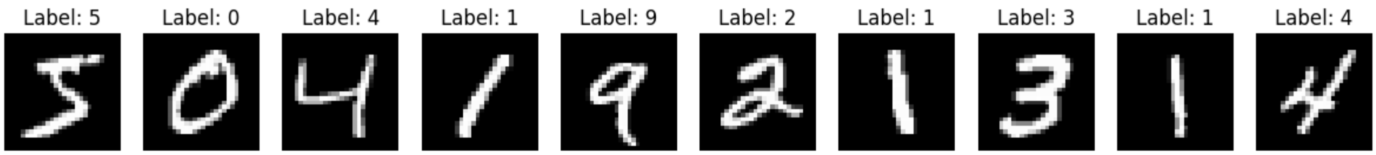
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
```

```
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader  # Import DataLoader
import matplotlib.pyplot as plt

# Define image transformations (normalize pixel values)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Download and load the dataset
train_dataset = torchvision.datasets.MNIST(root="./data", train=True, transform=transform, download=True)
val_dataset = torchvision.datasets.MNIST(root="./data", train=False, transform=transform, download=True)

# Create DataLoaders
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Print dataset size to confirm loading
print(f"Train dataset size: {len(train_dataset)}, Validation dataset size: {len(val_dataset)}")
```

```
Train dataset size: 60000, Validation dataset size: 10000
```

```
# Function to display images
def show_images(dataset, num_images=10):
    fig, axes = plt.subplots(1, num_images, figsize=(15, 2))
    for i in range(num_images):
        image, label = dataset[i]
```

```python
        axes[i].imshow(image.squeeze(), cmap="gray")
        axes[i].axis("off")
        axes[i].set_title(f"Label: {label}")
    plt.show()

show_images(train_dataset)
```




```python
import numpy as np

# Function to add noise to images
def forward_diffusion(x, t, noise_schedule):
    noise = torch.randn_like(x)  # Generate Gaussian noise
    alpha_t = noise_schedule[t].view(-1, 1, 1, 1)  # Adjust shape
    noisy_x = alpha_t * x + (1 - alpha_t) * noise  # Blend original image with noise
    return noisy_x


def reverse_diffusion(noisy_x, model, t, noise_schedule):
    predicted_x = model(noisy_x, t)  # Predict denoised image
    alpha_t = noise_schedule[t].view(-1, 1, 1, 1)
    denoised_x = (noisy_x - (1 - alpha_t) * predicted_x) / alpha_t  # Reverse noise addition
    return denoised_x


import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Define a basic U-Net for diffusion models
class UNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 64, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(128, 64, 2, stride=2),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 1, 2, stride=2),
            nn.Sigmoid()
        )

    def forward(self, x, t):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded  # Predict denoised image

# Initialize the model
model = UNet()


def train(model, train_loader, noise_schedule, epochs=5, lr=0.001):
    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_fn = nn.MSELoss()

    model.train()  # Set model to training mode
    for epoch in range(epochs):
        total_loss = 0
        for images, _ in train_loader:
            images = images.unsqueeze(1).to(device)  # Ensure proper shape (batch_size, 1, height, width)
            images = images.to(torch.float32)  # Convert tensor to float32 for consistency

            # ✅ Random diffusion timestep
```

```
            t = torch.randint(0, len(noise_schedule), (images.size(0),), device=device)

            # ✅ Forward diffusion (adding noise)
            noisy_images = forward_diffusion(images, t, noise_schedule)

            # ✅ Fix: Ensure correct shape before passing to the model
            noisy_images = noisy_images.squeeze(1)  # Remove extra dimension

            # ✅ Reverse diffusion prediction (denoising)
            denoised_images = model(noisy_images, t)

            # ✅ Compute loss
            loss = loss_fn(denoised_images, images)
            total_loss += loss.item()

            # ✅ Backpropagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        print(f"Epoch {epoch+1}, Loss: {total_loss / len(train_loader)}")


# Generate and visualize images after training
def generate_images(model, dataset, noise_schedule):
    fig, axes = plt.subplots(1, 5, figsize=(15, 3))

    for i in range(5):
        image, _ = dataset[i]

        # Ensure correct shape for visualization
        image = image.squeeze(0)  # Convert from (1, 28, 28) to (28, 28)

        noisy_image = forward_diffusion(image.unsqueeze(0), torch.tensor([50]), noise_schedule).squeeze(0)
        denoised_image = reverse_diffusion(noisy_image, model, torch.tensor([50]), noise_schedule).squeeze(0)

        # Convert tensor to numpy and ensure shape compatibility for imshow
        axes[i].imshow(denoised_image.detach().cpu().numpy().squeeze(), cmap="gray")
        axes[i].axis("off")

    plt.show()

# Define noise schedule
noise_schedule = torch.linspace(0.02, 1.0, steps=100)

# Call function after defining variables
generate_images(model, val_dataset, noise_schedule)
```