Step 1: Import Required Libraries

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
```

Step 2: Define Transformations & Load the MNIST Dataset

In [2]:
```
# Transform: Convert to tensor and normalize
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))  # mean and std of MNIST
])

# Load datasets
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, transfor
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, transfor

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1000, shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz (htt
p://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz)
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-
ubyte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ub
yte.gz)
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-
ubyte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ub
yte.gz) to ./data\MNIST\raw\train-images-idx3-ubyte.gz

                                                  9913344/? [00:02<00:00, 6204128.60it/s]

Extracting ./data\MNIST\raw\train-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz (htt
p://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz)
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-
ubyte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ub
yte.gz)
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-
ubyte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ub
yte.gz) to ./data\MNIST\raw\train-labels-idx1-ubyte.gz

                                                  29696/? [00:00<00:00, 4865.64it/s]

Extracting ./data\MNIST\raw\train-labels-idx1-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz (htt
p://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz)
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-u
byte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyt
e.gz)
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-u
byte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyt
e.gz) to ./data\MNIST\raw\t10k-images-idx3-ubyte.gz
```

```
1649664/? [00:01<00:00, 1145133.00it/s]

Extracting ./data\MNIST\raw\t10k-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz (http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz)
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz)
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz (https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz) to ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz

5120/? [00:00<00:00, 115724.27it/s]

Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw
```

Step 3: Define the CNN Model

In [10]:

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.25)

        # Create dummy input to calculate flattened size
        self._to_linear = None
        self._get_flattened_size()

        self.fc1 = nn.Linear(self._to_linear, 128)
        self.fc2 = nn.Linear(128, 10)

    def _get_flattened_size(self):
        x = torch.randn(1, 1, 28, 28)
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        self._to_linear = x.view(1, -1).shape[1]

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.fc2(x)
        return x
```

Step 4: Instantiate Model, Loss Function, and Optimizer

In [11]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Step 5: Train the Model

In [17]:
```
num_epochs = 5

train_losses = []
train_accuracies = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

        # Calculate accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    epoch_acc = 100 * correct / total

    # Store for plotting
    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_acc)

    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.4f}, Accuracy: {epoch_acc:.2f}
```

```
Epoch 1, Loss: 0.0069, Accuracy: 99.78%
Epoch 2, Loss: 0.0058, Accuracy: 99.78%
Epoch 3, Loss: 0.0074, Accuracy: 99.77%
Epoch 4, Loss: 0.0053, Accuracy: 99.81%
Epoch 5, Loss: 0.0054, Accuracy: 99.80%
```

In [15]:
```

```

```
Test Accuracy: 99.23%
```

In [19]:
```
# Step 6: Evaluate the Model
model.eval()
test_correct = 0
test_total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()

test_accuracy = 100 * test_correct / test_total
print(f'Test Accuracy: {test_accuracy:.2f}%')
```

```
Test Accuracy: 99.16%
```

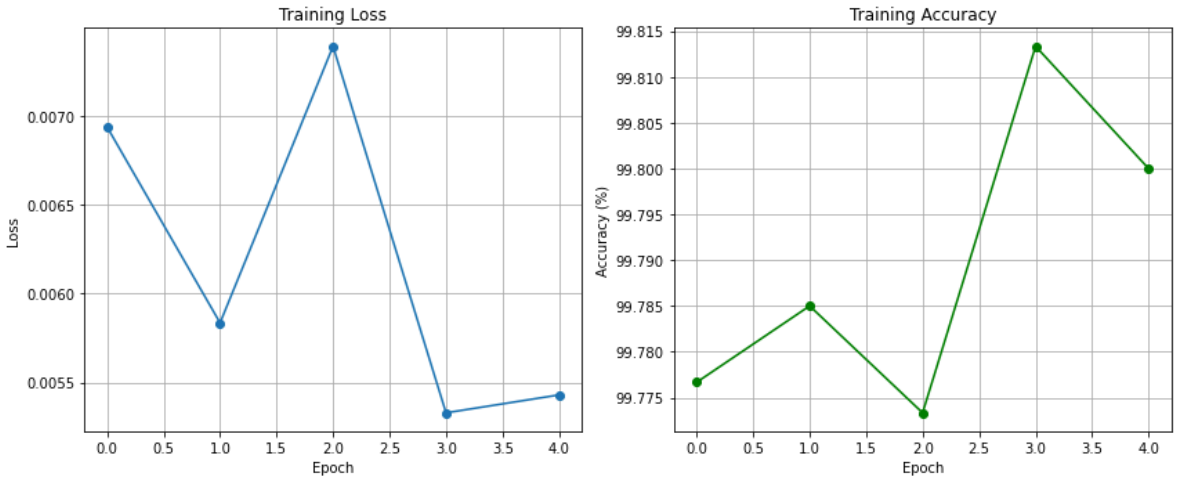
Step 7: Visualization of Training Results

In [20]:
```
# Step 7: Visualization of Training Results
plt.figure(figsize=(12, 5))

# Plot training loss
plt.subplot(1, 2, 1)
plt.plot(train_losses, marker='o')
plt.title("Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)

# Plot training accuracy
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, marker='o', color='green')
plt.title("Training Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.grid(True)

plt.tight_layout()
```



In [ ]: