```
from transformers import pipeline

# Initialize QA pipeline with default model
qa_pipeline = pipeline("question-answering")

# Context and Question
context = "The first mechanical computer was invented by Charles Babbage in the 19th century."
question = "Who invented the first mechanical computer?"

# Perform QA
result = qa_pipeline(question=question, context=context)

print("Task 1 Result:")
print(result)
```

```
No model was supplied, defaulted to distilbert/distilbert-base-cased-distilled-squad and revision 564e9b5 (https://huggi
Using a pipeline without specifying a model name and revision in production is not recommended.
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Device set to use cpu
Task 1 Result:
{'score': 0.995958149433136, 'start': 46, 'end': 61, 'answer': 'Charles Babbage'}
```

```
# Use custom pretrained QA model
qa_pipeline_custom = pipeline("question-answering", model="deepset/roberta-base-squad2")

# Perform QA with custom model
result_custom = qa_pipeline_custom(question=question, context=context)

print("Task 2 Result:")
print(result_custom)
```

```
config.json: 100%                                 571/571 [00:00<00:00, 31.2kB/s]
model.safetensors: 100%                                496M/496M [00:09<00:00, 88.9MB/s]
tokenizer_config.json: 100%                              79.0/79.0 [00:00<00:00, 3.67kB/s]
vocab.json: 100%                                  899k/899k [00:00<00:00, 5.43MB/s]
merges.txt: 100%                                  456k/456k [00:00<00:00, 7.44MB/s]
special_tokens_map.json: 100%                              772/772 [00:00<00:00, 59.4kB/s]
Device set to use cpu
Task 2 Result:
{'score': 0.9894621968269348, 'start': 46, 'end': 61, 'answer': 'Charles Babbage'}
```

```
from transformers import pipeline

# Use custom pretrained QA model
qa_pipeline_custom = pipeline("question-answering", model="deepset/roberta-base-squad2")

# Your custom context
my_context = (
    "edison invented light bulb "
    "light bulb runs with electricity"
)

# Two questions
question1 = "Who invented light bulb?"
question2 = "What does light bulb runs with?"

# Run QA on each question
answer1 = qa_pipeline_custom(question=question1, context=my_context)
answer2 = qa_pipeline_custom(question=question2, context=my_context)

print("Task 3 Result - Question 1:")
print(answer1)

print("Task 3 Result - Question 2:")
print(answer2)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100%                                   571/571 [00:00<00:00, 29.7kB/s]
model.safetensors: 100%                             496M/496M [00:06<00:00, 21.3MB/s]
tokenizer_config.json: 100%                         79.0/79.0 [00:00<00:00, 6.80kB/s]
vocab.json: 100%                                    899k/899k [00:00<00:00, 5.88MB/s]
merges.txt: 100%                                    456k/456k [00:00<00:00, 6.36MB/s]
special_tokens_map.json: 100%                       772/772 [00:00<00:00, 38.3kB/s]
Device set to use cpu
Task 3 Result - Question 1:
{'score': 0.9080078601837158, 'start': 0, 'end': 6, 'answer': 'edison'}
Task 3 Result - Question 2:
{'score': 0.9728672504425049, 'start': 48, 'end': 59, 'answer': 'electricity'}
```

```python
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
import numpy as np
```

```python
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])  # Scale images to [-1, 1]
])

mnist = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
dataloader = torch.utils.data.DataLoader(mnist, batch_size=128, shuffle=True)
```

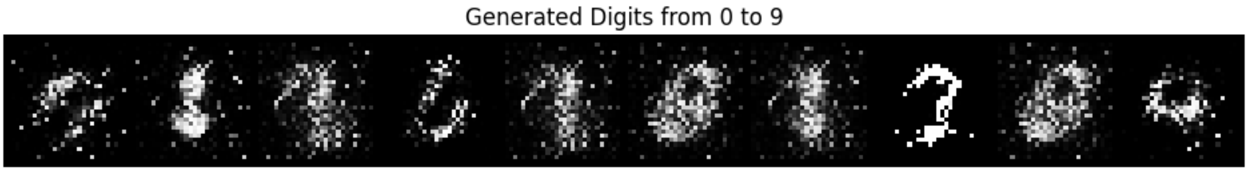
```
100%|██████████| 9.91M/9.91M [00:00<00:00, 15.9MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 500kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 3.98MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 6.27MB/s]
```

```python
class Generator(nn.Module):
    def __init__(self, noise_dim, label_dim, img_shape):
        super().__init__()
        self.label_embed = nn.Embedding(10, label_dim)
        self.model = nn.Sequential(
            nn.Linear(noise_dim + label_dim, 128),
            nn.ReLU(True),
            nn.Linear(128, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(True),
            nn.Linear(512, int(np.prod(img_shape))),
            nn.Tanh()
        )
        self.img_shape = img_shape

    def forward(self, noise, labels):
        x = torch.cat((noise, self.label_embed(labels)), dim=1)
        img = self.model(x)
        return img.view(img.size(0), *self.img_shape)
```

```python
class Discriminator(nn.Module):
    def __init__(self, label_dim, img_shape):
        super().__init__()
        self.label_embed = nn.Embedding(10, label_dim)
        self.model = nn.Sequential(
            nn.Linear(np.prod(img_shape) + label_dim, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, img, labels):
        x = torch.cat((img.view(img.size(0), -1), self.label_embed(labels)), dim=1)
        return self.model(x)



device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
img_shape = (1, 28, 28)
noise_dim = 100
label_dim = 10

G = Generator(noise_dim, label_dim, img_shape).to(device)
D = Discriminator(label_dim, img_shape).to(device)

criterion = nn.BCELoss()
optimizer_G = torch.optim.Adam(G.parameters(), lr=0.0002)
optimizer_D = torch.optim.Adam(D.parameters(), lr=0.0002)

for epoch in range(10):
    for imgs, labels in dataloader:
        batch_size = imgs.size(0)
        real_imgs = imgs.to(device)
        labels = labels.to(device)

        # Real and fake labels
        valid = torch.ones(batch_size, 1).to(device)
        fake = torch.zeros(batch_size, 1).to(device)

        # Train Generator
        optimizer_G.zero_grad()
        z = torch.randn(batch_size, noise_dim).to(device)
        gen_labels = torch.randint(0, 10, (batch_size,), device=device)
        gen_imgs = G(z, gen_labels)
        g_loss = criterion(D(gen_imgs, gen_labels), valid)
        g_loss.backward()
        optimizer_G.step()

        # Train Discriminator
        optimizer_D.zero_grad()
        real_loss = criterion(D(real_imgs, labels), valid)
        fake_loss = criterion(D(gen_imgs.detach(), gen_labels), fake)
        d_loss = real_loss + fake_loss
        d_loss.backward()
        optimizer_D.step()

    print(f"Epoch {epoch+1} | D Loss: {d_loss.item():.4f} | G Loss: {g_loss.item():.4f}")
```

```
Epoch 1 | D Loss: 0.0091 | G Loss: 12.0936
Epoch 2 | D Loss: 0.0470 | G Loss: 9.4788
Epoch 3 | D Loss: 0.2744 | G Loss: 4.8249
Epoch 4 | D Loss: 0.2639 | G Loss: 4.0875
Epoch 5 | D Loss: 0.2730 | G Loss: 6.5120
Epoch 6 | D Loss: 0.1832 | G Loss: 3.4900
Epoch 7 | D Loss: 0.1945 | G Loss: 2.8919
Epoch 8 | D Loss: 0.1654 | G Loss: 4.6466
Epoch 9 | D Loss: 0.1128 | G Loss: 4.5387
Epoch 10 | D Loss: 0.0895 | G Loss: 4.4215
```

```
def generate_digits_per_class(generator, device):
    generator.eval()
    z = torch.randn(10, noise_dim).to(device)
    labels = torch.arange(0, 10).to(device)
    gen_imgs = generator(z, labels)
    gen_imgs = gen_imgs.cpu().detach()
    grid = make_grid(gen_imgs, nrow=10, normalize=True)
    plt.figure(figsize=(12, 2))
    plt.imshow(np.transpose(grid, (1, 2, 0)))
    plt.axis('off')
    plt.title("Generated Digits from 0 to 9")
    plt.show()

generate_digits_per_class(G, device)
```


Generated Digits from 0 to 9