```
!pip install -q diffusers transformers accelerate safetensors

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os

dataset_dir = "/content/drive/MyDrive/genai_lab1_dataset"
os.makedirs(dataset_dir, exist_ok=True)
print("Saving images to:", dataset_dir)
```

```
Saving images to: /content/drive/MyDrive/genai_lab1_dataset
```

```
import torch
from diffusers import StableDiffusionPipeline

model_id = "runwayml/stable-diffusion-v1-5"  # popular SD model

device = "cuda" if torch.cuda.is_available() else "cpu"
pipe = StableDiffusionPipeline.from_pretrained(
    model_id,
    torch_dtype=torch.float16 if device == "cuda" else torch.float32
)
pipe = pipe.to(device)
```

```
Flax classes are deprecated and will be removed in Diffusers v1.0.0. W
Flax classes are deprecated and will be removed in Diffusers v1.0.0. W
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your sett
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to a
  warnings.warn(
model_index.json: 100%                                  541/541 [00:00<00:00, 59.7kB/s]
Fetching 15 files: 100%                                   15/15 [00:49<00:00,  2.51s/it]
merges.txt:       525k/? [00:00<00:00, 5.40MB/s]
config.json:      4.72k/? [00:00<00:00, 120kB/s]
special_tokens_map.json: 100%                           472/472 [00:00<00:00, 6.52kB/s]
preprocessor_config.json: 100%                          342/342 [00:00<00:00, 5.58kB/s]
config.json: 100%                                       617/617 [00:00<00:00, 7.43kB/s]
scheduler_config.json: 100%                             308/308 [00:00<00:00, 3.81kB/s]
text_encoder/model.safetensors: 100%                    492M/492M [00:48<00:00, 7.45MB/s]
safety_checker/model.safetensors: 100%                  1.22G/1.22G [00:47<00:00, 7.08MB/s]
tokenizer_config.json: 100%                             806/806 [00:00<00:00, 33.0kB/s]
config.json: 100%                                       743/743 [00:00<00:00, 22.2kB/s]
vocab.json:       1.06M/? [00:00<00:00, 12.2MB/s]
config.json: 100%                                       547/547 [00:00<00:00, 21.9kB/s]
unet/diffusion_pytorch_model.safetensors: 100%          3.44G/3.44G [00:48<00:00, 240MB/s]
vae/diffusion_pytorch_model.safetensors: 100%           335M/335M [00:20<00:00, 8.72MB/s]
Loading pipeline components...: 100%                      7/7 [00:23<00:00,  3.70s/it]
`torch_dtype` is deprecated! Use `dtype` instead!
```

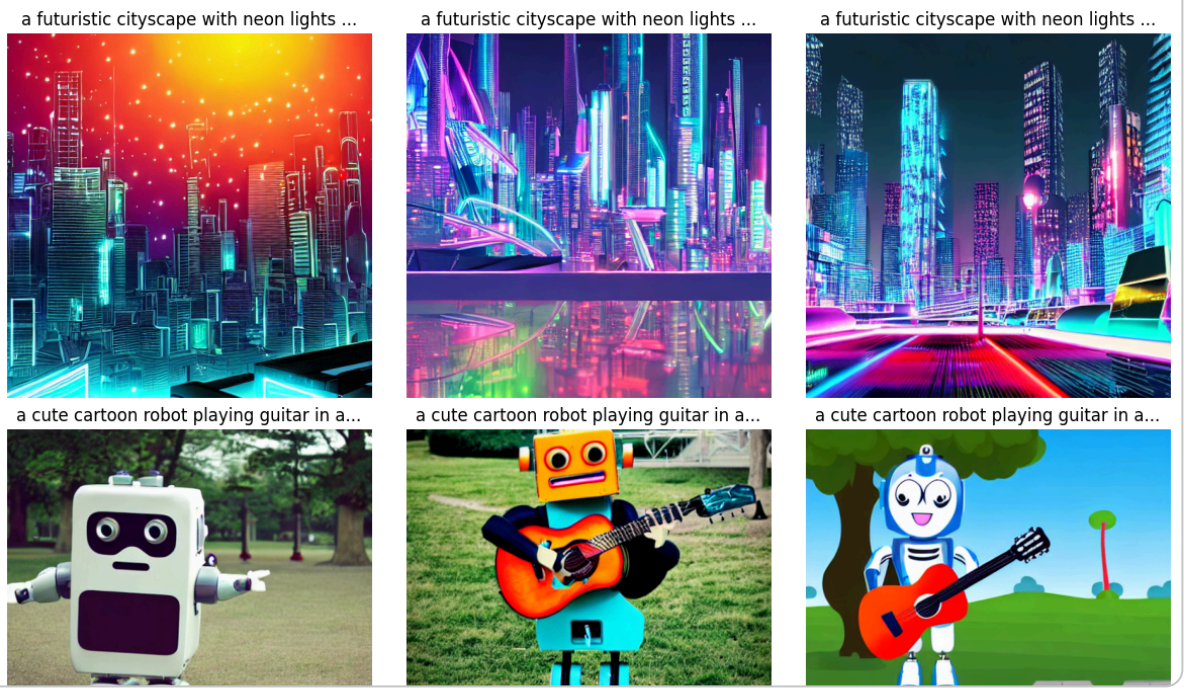
```
import uuid
from PIL import Image

prompts = [
    "a futuristic cityscape with neon lights at night, ultra detailed
    "a cute cartoon robot playing guitar in a park",
    "a realistic portrait of a dog wearing sunglasses, studio lightin
    "a fantasy castle on a floating island, bright colors, concept ar
    "a sports car racing on a mountain road at sunset"
]

images_per_prompt = 3
guidance_scale = 7.5
```

```
num_inference_steps = 30

all_paths = []

for prompt in prompts:
    print("Generating for prompt:", prompt)
    for i in range(images_per_prompt):
        image = pipe(
            prompt,
            guidance_scale=guidance_scale,
            num_inference_steps=num_inference_steps
        ).images[0]           # PIL Image [web:8]
        filename = f"{uuid.uuid4().hex}.png"
        save_path = os.path.join(dataset_dir, filename)
        image.save(save_path)
        all_paths.append((prompt, save_path))
        print("  Saved:", save_path)
```

```
Generating for prompt: a futuristic cityscape with neon lights at nigh
100%                                                30/30 [00:05<00:00,  7.63it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/cc92873d47e14683a2e
100%                                                30/30 [00:04<00:00,  7.60it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/5f2c8d5a457d438380e
100%                                                30/30 [00:04<00:00,  7.57it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/9ce63ee2a37a4715818
Generating for prompt: a cute cartoon robot playing guitar in a park
100%                                                30/30 [00:04<00:00,  7.46it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/0808670b0192456d961
100%                                                30/30 [00:04<00:00,  7.40it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/a4b95292e1a541aa8dd
100%                                                30/30 [00:04<00:00,  7.40it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/dbb1d21a5cd94688baf
Generating for prompt: a realistic portrait of a dog wearing sunglasse
100%                                                30/30 [00:04<00:00,  7.26it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/15e9a6a57c3146fc8e3
100%                                                30/30 [00:04<00:00,  7.26it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/461b2436d3134d9bab8
100%                                                30/30 [00:04<00:00,  7.23it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/a2ed5ba37d514b47a6c
Generating for prompt: a fantasy castle on a floating island, bright c
100%                                                30/30 [00:04<00:00,  7.13it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/ac5b7da3e1f3462f9c6
100%                                                30/30 [00:04<00:00,  7.07it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/db065b5588104edda1e
100%                                                30/30 [00:04<00:00,  7.05it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/885ef4fa6a964494894
Generating for prompt: a sports car racing on a mountain road at sunse
100%                                                30/30 [00:04<00:00,  7.00it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/8a8329abc2914bb6bfd
100%                                                30/30 [00:04<00:00,  6.87it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/56c7e67bf6644bc0a99
100%                                                30/30 [00:04<00:00,  6.93it/s]
  Saved: /content/drive/MyDrive/genai_lab1_dataset/9df18fbd4e2c4d39907
```

```
import matplotlib.pyplot as plt

n_show = min(6, len(all_paths))
plt.figure(figsize=(12, 8))

for i in range(n_show):
    prompt, path = all_paths[i]
    img = Image.open(path)
```

```
        plt.subplot(2, 3, i + 1)
        plt.imshow(img)
        plt.axis("off")
        plt.title(prompt[:40] + "..." if len(prompt) > 40 else prompt)

plt.tight_layout()
plt.show()
```




```
import os
import uuid
from PIL import Image

# Root folder for X-ray synthetic dataset
xray_root = "/content/drive/MyDrive/genai_xray_dataset"
os.makedirs(xray_root, exist_ok=True)

# Choose 4–5 diseases / classes
diseases = [
    "normal chest",
    "pneumonia",
    "tuberculosis",
    "lung cancer",
    "pulmonary edema"
]

images_per_disease = 5  # 4–5 images per disease
print("Saving X-ray style images under:", xray_root)
```

```
Saving X-ray style images under: /content/drive/MyDrive/genai_xray_dat
```

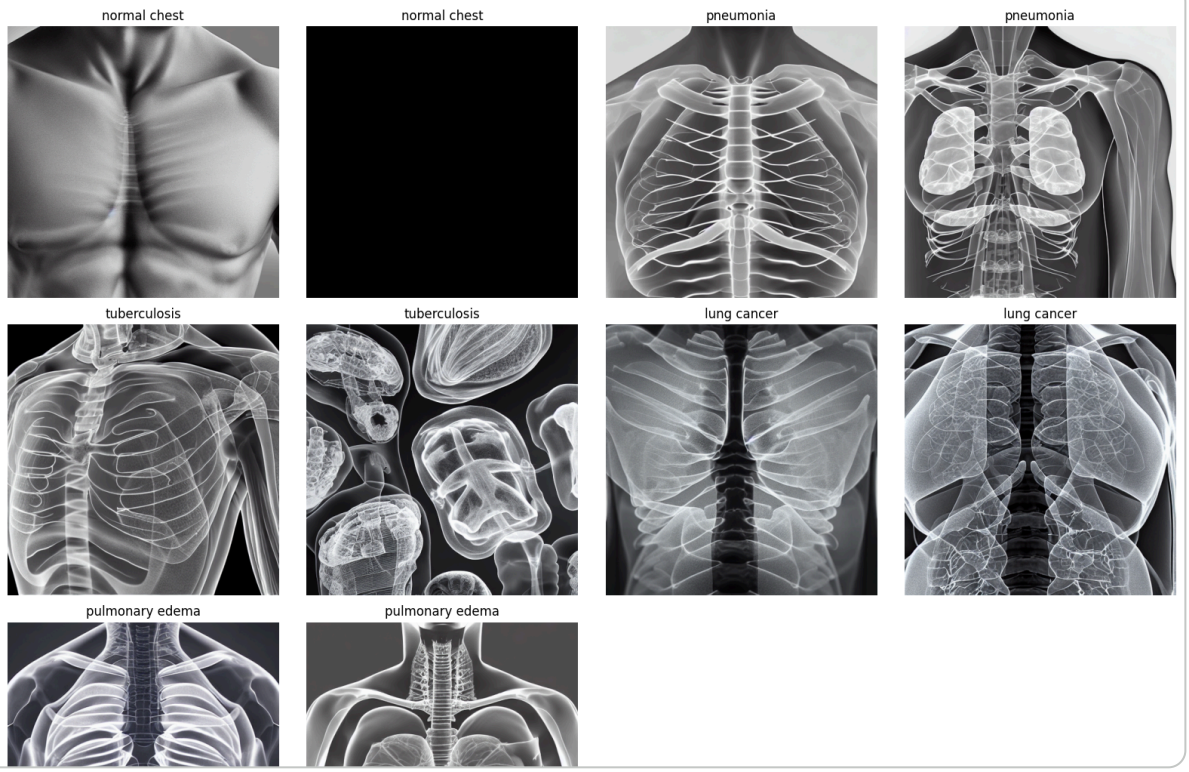
```
guidance_scale = 7.5
num_inference_steps = 30
```

```
xray_paths = []

for disease in diseases:
    disease_slug = disease.replace(" ", "_").lower()
    disease_dir = os.path.join(xray_root, disease_slug)
    os.makedirs(disease_dir, exist_ok=True)

    print(f"\n=== Generating for disease: {disease} ===")

    for i in range(images_per_disease):
        prompt = (
            f"high resolution grayscale chest X-ray image, "
            f"frontal view, {disease}, medical imaging, "
            f"clean background, realistic radiology style"
        )

        image = pipe(
            prompt,
            guidance_scale=guidance_scale,
            num_inference_steps=num_inference_steps
        ).images[0]

        filename = f"{disease_slug}_{uuid.uuid4().hex}.png"
        save_path = os.path.join(disease_dir, filename)
        image.save(save_path)
        xray_paths.append((disease, prompt, save_path))
        print("  Saved:", save_path)
```

```
=== Generating for disease: normal chest ===
100%                                                  30/30 [00:04<00:00,  6.83it/s]
Potential NSFW content was detected in one or more images. A black im
  Saved: /content/drive/MyDrive/genai_xray_dataset/normal_chest/norma
100%                                                  30/30 [00:05<00:00,  4.88it/s]
  Saved: /content/drive/MyDrive/genai_xray_dataset/normal_chest/norma
100%                                                  30/30 [00:05<00:00,  6.62it/s]
  Saved: /content/drive/MyDrive/genai_xray_dataset/normal_chest/norma
100%                                                  30/30 [00:04<00:00,  6.68it/s]
  Saved: /content/drive/MyDrive/genai_xray_dataset/normal_chest/norma
100%                                                  30/30 [00:04<00:00,  6.69it/s]
  Saved: /content/drive/MyDrive/genai_xray_dataset/normal_chest/norma

=== Generating for disease: pneumonia ===
100%                                                  30/30 [00:04<00:00,  6.72it/s]
  Saved: /content/drive/MyDrive/genai_xray_dataset/pneumonia/pneumoni
100%                                                  30/30 [00:04<00:00,  6.73it/s]
  Saved: /content/drive/MyDrive/genai_xray_dataset/pneumonia/pneumoni
```

```
import random
import matplotlib.pyplot as plt

# Take up to 2 random images per disease for display
samples_per_disease = 2
to_show = []

for disease in diseases:
    items = [rec for rec in xray_paths if rec[0] == disease]
    random.shuffle(items)
    to_show.extend(items[:samples_per_disease])

n_show = len(to_show)
cols = 4
rows = (n_show + cols - 1) // cols

plt.figure(figsize=(4 * cols, 4 * rows))

for i, (disease, prompt, path) in enumerate(to_show, start=1):
    img = Image.open(path)
    plt.subplot(rows, cols, i)
    plt.imshow(img, cmap="gray")
    plt.axis("off")
    plt.title(disease)

plt.tight_layout()
plt.show()
```

```
  Saved: /content/drive/MyDrive/genai_xray_dataset/lung_cancer/lung_c
100%                                                  30/30 [00:04<00:00,  6.76it/s]
  Saved: /content/drive/MyDrive/genai_xray_dataset/lung_cancer/lung_c
```




```
!pip install -q torch torchvision torchaudio

import os
from PIL import Image
from sklearn.metrics import accuracy_score, f1_score
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import torch
from torch.utils.data import DataLoader, Dataset
from torchvision import datasets, transforms
import os
from sklearn.model_selection import train_test_split

# Custom dataset class to handle image paths and labels after splitti
class CustomImageDataset(Dataset):
    def __init__(self, image_paths, labels, transform=None):
        self.image_paths = image_paths
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)
```

```
    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        # Use default_loader from torchvision.datasets.folder to load
        image = datasets.folder.default_loader(img_path)
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label

data_dir = "/content/drive/MyDrive/genai_xray_dataset"

batch_size = 8
img_size = 224

train_tfms = transforms.Compose([
    transforms.Resize((img_size, img_size)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

test_tfms = transforms.Compose([
    transforms.Resize((img_size, img_size)),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

# --- Stratified split implementation ---
# First, get all image paths and their corresponding labels
all_image_paths = []
all_labels = []

# Load the dataset once to get class names and map them to indices
full_raw_dataset = datasets.ImageFolder(root=data_dir)
class_names = full_raw_dataset.classes
class_to_idx = full_raw_dataset.class_to_idx

for class_name in class_names:
    class_path = os.path.join(data_dir, class_name)
    for img_name in os.listdir(class_path):
        if img_name.endswith(('.png', '.jpg', '.jpeg')):
            all_image_paths.append(os.path.join(class_path, img_name)
            all_labels.append(class_to_idx[class_name])

# Perform a stratified split
X_train, X_val, y_train, y_val = train_test_split(
    all_image_paths, all_labels, test_size=0.2, random_state=42, stra
)

# Create custom datasets for train and validation using the split pat
train_dataset = CustomImageDataset(X_train, y_train, transform=train_
val_dataset = CustomImageDataset(X_val, y_val, transform=test_tfms) #
```

```
print("Classes:", class_names)
print(f"Total training images: {len(train_dataset)}")
print(f"Total validation images: {len(val_dataset)}")

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuff
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=F
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
Classes: ['lung_cancer', 'normal_chest', 'pneumonia', 'pulmonary_edema
Total training images: 40
Total validation images: 10
'cuda'
```

```
import torch.nn as nn
from sklearn.metrics import accuracy_score, f1_score

def train_one_model(model, model_name, num_epochs=3, lr=1e-4):
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        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() * images.size(0)

        epoch_loss = running_loss / len(train_loader.dataset)
        print(f"[{model_name}] Epoch {epoch+1}/{num_epochs}, Train lo

    # evaluation
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            preds = outputs.argmax(dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    acc = accuracy_score(all_labels, all_preds)
```