Sample Python code for training a Small LLM using Hugging Face

# Code generated with assistance from ChatGPT (OpenAI, 2025)

```
pip install transformers datasets torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer, TrainingArguments, TextDataset, DataCollatorForLanguageModeling

# Load pretrained model and tokenizer
model_name = "gpt2"
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)

# Load and tokenize your dataset
def load_dataset(file_path, tokenizer, block_size=128):
    return TextDataset(
        tokenizer=tokenizer,
        file_path=file_path,
        block_size=block_size
    )
# Load data collator
data_collator = DataCollatorForLanguageModeling(
    tokenizer=tokenizer,
    mlm=False
)
# Replace this with your text file path
train_dataset = load_dataset("train.txt", tokenizer)

# Define training arguments
training_args = TrainingArguments(
    output_dir="./gpt2-finetuned",
    overwrite_output_dir=True,
    num_train_epochs=3,
    per_device_train_batch_size=2,
    save_steps=500,
    save_total_limit=2,
    prediction_loss_only=True,
    logging_steps=100
)
# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=train_dataset
```

```
)

# Start training
trainer.train()
```

The following python codes helps to generate image from the MNIST dataset (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)

import torch
import torch.nn as kk
import torch.nn.functional as K
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import torch.optim as om

class VAE(kk.Module):
    def __init__(self, input_dim=889, hidden_dim=200, latent_dim=25):
        super(VAE, self).__init__()
        self.fc1 = kk.Linear(input_dim, hidden_dim)
        self.fc_mu = kk.Linear(hidden_dim, latent_dim)
        self.fc_logvar = kk.Linear(hidden_dim, latent_dim)
        self.fc3 = kk.Linear(latent_dim, hidden_dim)
        self.fc4 = kk.Linear(hidden_dim, input_dim)

    def encode(self, x):
        h1 = K.relu(self.fc1(x))
        mu = self.fc_mu(h1)
        logvar = self.fc_logvar(h1)
        return mu, logvar

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

    def decode(self, z):
        h3 = K.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h3))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 889))
        z = self.reparameterize(mu, logvar)
        recon_x = self.decode(z)
        return recon_x, mu, logvar


def vae_loss(recon_x, x, mu, logvar):
    BCE = K.binary_cross_entropy(recon_x, x.view(-1,889), reduction='sum')
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD

transform = transforms.ToTensor()
train_dataset = datasets.MNIST('./data', train=True, download=True, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=148, shuffle=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = VAE().to(device)
optimizer = om.Adam(model.parameters(), lr=1e-3)
```

```
num_epochs = 12
model.train()
for epoch in range(num_epochs):
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(data)
        loss = vae_loss(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    print(f'Epoch {epoch + 1}, Loss: {train_loss / len(train_loader.dataset):.4f}')

import matplotlib.pyplot as plt

model.eval()
with torch.no_grad():
    z = torch.randn(164,12).to(device)
    samples = model.decode(z).cpu().view(-1, 1, 38, 38)

    grid_img = torchvision.utils.make_grid(samples, nrow=8)
    plt.figure(figsize=(8, 8))
    plt.imshow(grid_img.permute(1, 2, 0).squeeze())
    plt.axis('off')
    plt.show()
```

The following Python code  EMB models that assigns low energy to real 2D points near the origin and high energy to other points: (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)

import torch
import torch.nn as nn
import matplotlib.pyplot as plt

# Simple energy function using a small neural network
class EnergyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(2, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

    def forward(self, x):
        return self.net(x).squeeze()

# Sample synthetic data (points near origin)
def sample_data(n=1000):
    return torch.randn(n, 2) * 0.5

# Langevin dynamics to sample from the model
def langevin_sample(model, steps=30, lr=0.1, n_samples=100):
    x = torch.randn(n_samples, 2).requires_grad_()
    for _ in range(steps):
        energy = model(x)
        grad = torch.autograd.grad(energy.sum(), x)[0]
        x.data -= lr * grad + 0.01 * torch.randn_like(x)
    return x.detach()

# Training loop
def train(model, data, epochs=1000, lr=1e-3):
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    for epoch in range(epochs):
        # Positive samples (real data)
        pos_data = data
        pos_energy = model(pos_data).mean()
```

```
    # Negative samples (via Langevin dynamics)
    neg_data = langevin_sample(model, n_samples=pos_data.size(0))
    neg_energy = model(neg_data).mean()

    loss = pos_energy - neg_energy
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 100 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")

# Run it
model = EnergyModel()
data = sample_data()
train(model, data)

# Visualize energy landscape
import numpy as np

xx, yy = torch.meshgrid(torch.linspace(-3, 3, 100), torch.linspace(-3, 3, 100))
grid = torch.stack([xx.flatten(), yy.flatten()], dim=-1)
energy = model(grid).detach().numpy().reshape(100, 100)

plt.contourf(xx, yy, energy, levels=50, cmap='viridis')
plt.colorbar(label='Energy')
plt.title("Learned Energy Landscape")
plt.show()
```

The following Python code helps us to generate flow based model using Real-valued Non-Volume Preserving transformations (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)
pip install torch matplotlib numpy
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt

# Simple MLP for scaling and translation
class MLP(nn.Module):
  def __init__(self, input_dim, hidden_dim):
    super(MLP, self).__init__()
    self.net = nn.Sequential(
      nn.Linear(input_dim, hidden_dim),
      nn.ReLU(),
      nn.Linear(hidden_dim, input_dim)
    )

  def forward(self, x):
    return self.net(x)

# RealNVP coupling layer
class CouplingLayer(nn.Module):
  def __init__(self, input_dim, hidden_dim, mask):
    super().__init__()
    self.mask = mask
    self.scale_net = MLP(input_dim, hidden_dim)
    self.translate_net = MLP(input_dim, hidden_dim)

  def forward(self, x):
    x_masked = x * self.mask
    scale = self.scale_net(x_masked) * (1 - self.mask)
    translate = self.translate_net(x_masked) * (1 - self.mask)
    z = x_masked + (1 - self.mask) * (x * torch.exp(scale) + translate)
    log_det_jacobian = torch.sum(scale, dim=1)
    return z, log_det_jacobian

  def inverse(self, z):
    z_masked = z * self.mask
    scale = self.scale_net(z_masked) * (1 - self.mask)
    translate = self.translate_net(z_masked) * (1 - self.mask)
    x = z_masked + (1 - self.mask) * ((z - translate) * torch.exp(-scale))
```

```
        return x

# RealNVP model with multiple layers
class RealNVP(nn.Module):
    def __init__(self, input_dim=2, hidden_dim=256, num_layers=6):
        super().__init__()
        self.layers = nn.ModuleList()
        for i in range(num_layers):
            mask = self._get_mask(i, input_dim)
            self.layers.append(CouplingLayer(input_dim, hidden_dim, mask))

    def _get_mask(self, layer, dim):
        mask = torch.zeros(dim)
        mask[layer % dim::2] = 1
        return mask

    def forward(self, x):
        log_det_jacobian = 0
        for layer in self.layers:
            x, log_det = layer(x)
            log_det_jacobian += log_det
        return x, log_det_jacobian

    def inverse(self, z):
        for layer in reversed(self.layers):
            z = layer.inverse(z)
        return z
# Sampling from latent space
with torch.no_grad():
    z = torch.randn(1000, 2)
    samples = model.inverse(z).numpy()

plt.figure(figsize=(6, 6))
plt.scatter(samples[:, 0], samples[:, 1], s=5, alpha=0.6)
plt.title("Samples from Flow-Based Model")
plt.axis("equal")
plt.show()
```

The following python codes helps us to use Generative model Variational Autoencoder (VAE) to model the environment, which is then used to train an RL agent generate handwritten digits using GAN (The code is generated using ChatGPT (Open Ai, 2025)_

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)

pip install gym torch torchvision stable-baselines3
import gym
import torch
import torch.nn as nn
import torch.optim as optim
from stable_baselines3 import PPO
import numpy as np

# VAE for encoding observations
class VAE(nn.Module):
    def __init__(self, obs_dim, latent_dim=16):
        super(VAE, self).__init__()
        self.fc1 = nn.Linear(obs_dim, 32)
        self.fc21 = nn.Linear(32, latent_dim)  # mean
        self.fc22 = nn.Linear(32, latent_dim)  # log variance
        self.fc3 = nn.Linear(latent_dim, 32)
        self.fc4 = nn.Linear(32, obs_dim)

    def encode(self, x):
        h1 = torch.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

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

    def decode(self, z):
        h3 = torch.relu(self.fc3(z))
        return self.fc4(h3)

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar
# Loss for VAE
def vae_loss(recon_x, x, mu, logvar):
    recon_loss = nn.functional.mse_loss(recon_x, x)
    kl = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
```

```
    return recon_loss + kl * 0.001
# Collect data for VAE training
def collect_data(env, episodes=100):
    data = []
    for _ in range(episodes):
        obs = env.reset()
        done = False
        while not done:
            data.append(obs)
            action = env.action_space.sample()
            obs, _, done, _ = env.step(action)
    return torch.tensor(data, dtype=torch.float32)
# Train the VAE
def train_vae(vae, data, epochs=50):
    optimizer = optim.Adam(vae.parameters(), lr=1e-3)
    for epoch in range(epochs):
        recon_batch, mu, logvar = vae(data)
        loss = vae_loss(recon_batch, data, mu, logvar)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch % 10 == 0:
            print(f'Epoch {epoch}, Loss: {loss.item():.4f}')
# Main
env = gym.make('CartPole-v1')
obs_dim = env.observation_space.shape[0]
vae = VAE(obs_dim)
# Step 1: Collect data & train VAE
dataset = collect_data(env, episodes=200)
train_vae(vae, dataset)

# Step 2: Replace env with encoded observations (simulate using latent space)
class VAECartPoleWrapper(gym.ObservationWrapper):
    def __init__(self, env, vae):
        super().__init__(env)
        self.vae = vae
        self.observation_space = gym.spaces.Box(low=-np.inf, high=np.inf, shape=(16,),
dtype=np.float32)

    def observation(self, obs):
        obs_tensor = torch.tensor(obs, dtype=torch.float32)
        mu, _ = self.vae.encode(obs_tensor)
        return mu.detach().numpy()
vae_env = VAECartPoleWrapper(env, vae)
# Step 3: Train RL agent in latent space
```

```
model = PPO("MlpPolicy", vae_env, verbose=1)
model.learn(total_timesteps=10000)
model.save("ppo_grl_model")
```

The following python codes helps us to generate handwritten digits using GAN (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)
pip install torch torchvision matplotlib
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Hyperparameters
latent_dim = 100
batch_size = 64
lr = 0.0002
epochs = 50

# MNIST data
transform = transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize([0.5], [0.5])  # Normalize to [-1, 1]
])

dataloader = DataLoader(
   datasets.MNIST('.', train=True, download=True, transform=transform),
   batch_size=batch_size, shuffle=True
)

# Generator model
class Generator(nn.Module):
   def __init__(self):
      super().__init__()
      self.model = nn.Sequential(
         nn.Linear(latent_dim, 256),
         nn.ReLU(),
         nn.Linear(256, 512),
         nn.ReLU(),
         nn.Linear(512, 784),
         nn.Tanh()
      )
```

```
    def forward(self, z):
        img = self.model(z)
        return img.view(z.size(0), 1, 28, 28)

# Discriminator model
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, img):
        return self.model(img)

# Initialize models
generator = Generator().to(device)
discriminator = Discriminator().to(device)

# Loss and optimizers
criterion = nn.BCELoss()
optimizer_G = optim.Adam(generator.parameters(), lr=lr)
optimizer_D = optim.Adam(discriminator.parameters(), lr=lr)

# Training loop
for epoch in range(epochs):
    for i, (real_imgs, _) in enumerate(dataloader):
        real_imgs = real_imgs.to(device)
        real_labels = torch.ones(real_imgs.size(0), 1).to(device)
        fake_labels = torch.zeros(real_imgs.size(0), 1).to(device)

        # Train Discriminator
        optimizer_D.zero_grad()
        z = torch.randn(real_imgs.size(0), latent_dim).to(device)
        fake_imgs = generator(z)
        real_loss = criterion(discriminator(real_imgs), real_labels)
        fake_loss = criterion(discriminator(fake_imgs.detach()), fake_labels)
        d_loss = real_loss + fake_loss
        d_loss.backward()
```

```
    optimizer_D.step()

    # Train Generator
    optimizer_G.zero_grad()
    z = torch.randn(real_imgs.size(0), latent_dim).to(device)
    fake_imgs = generator(z)
    g_loss = criterion(discriminator(fake_imgs), real_labels)
    g_loss.backward()
    optimizer_G.step()

print(f"Epoch [{epoch+1}/{epochs}]  D Loss: {d_loss.item():.4f}  G Loss: {g_loss.item():.4f}")

# Visualize generated images every few epochs
if (epoch + 1) % 10 == 0:
    with torch.no_grad():
        test_z = torch.randn(16, latent_dim).to(device)
        generated = generator(test_z).cpu()
        grid = torch.cat([img.squeeze() for img in generated], dim=1)
        plt.imshow(grid.detach().numpy(), cmap='gray')
        plt.title(f"Epoch {epoch+1}")
        plt.axis('off')
        plt.show()
```

The following Python code helps us to understand the RAG process (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)

pip install faiss-cpu sentence-transformers openai
  Code: Simple RAG with FAISS + GPT

import faiss
import openai
from sentence_transformers import SentenceTransformer
import numpy as np

# Setup
openai.api_key = "your-openai-key"
embed_model = SentenceTransformer("all-MiniLM-L6-v2")

# Sample documents
docs = [
   "The Eiffel Tower is located in Paris.",
   "The Great Wall of China is visible from space.",
   "Python is a popular programming language for AI.",
   "The capital of Japan is Tokyo."
]

# Embed and build FAISS index
doc_embeddings = embed_model.encode(docs)
dimension = doc_embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(np.array(doc_embeddings))

# User query
query = "Where is the Eiffel Tower?"
query_vec = embed_model.encode([query])

# Retrieve top-1 relevant doc
_, indices = index.search(np.array(query_vec), k=1)
retrieved = docs[indices[0][0]]

# Augment prompt and generate
prompt = f"Context: {retrieved}\n\nQuestion: {query}\nAnswer:"
response = openai.ChatCompletion.create(
   model="gpt-4",
   messages=[{"role": "user", "content": prompt}]
```

```
)
print(response['choices'][0]['message']['content'])
```

The following Python code helps us to generate videos based on the user input (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)
pip install diffusers transformers accelerate torch torchvision imageio

import torch
from diffusers import StableDiffusionPipeline, StableVideoDiffusionPipeline
from PIL import Image
import imageio

# Set up your prompt
prompt = "A peaceful mountain landscape at sunrise, animated in slow motion"

# Load text-to-image pipeline (Stable Diffusion)
text2img_pipe = StableDiffusionPipeline.from_pretrained(
   "runwayml/stable-diffusion-v1-5",
   torch_dtype=torch.float16
).to("cuda")

# Generate image from text
image = text2img_pipe(prompt).images[0]
image.save("generated_frame.png")

# Load image-to-video model
video_pipe = StableVideoDiffusionPipeline.from_pretrained(
   "stabilityai/stable-video-diffusion-img2vid-xt",
   torch_dtype=torch.float16
).to("cuda")

# Generate video frames
result = video_pipe(image, num_frames=14, decode_chunk_size=4)
frames = result.frames[0]  # List of PIL images

# Save as video
video_path = "output_video.mp4"
imageio.mimsave(video_path, frames, fps=8)

print(f"  Video saved to {video_path}")
```

The following Python code helps us to generate image based on the input given by the user (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)
pip install diffusers transformers torch
from diffusers import StableDiffusionPipeline
import torch
# Load the model (requires Hugging Face token)
pipe = StableDiffusionPipeline.from_pretrained(
   "CompVis/stable-diffusion-v1-4",
   torch_dtype=torch.float16,
   revision="fp16",
   use_auth_token=True
).to("cuda")  # use "cpu" if you don't have a GPU

# Generate image from prompt
prompt = "a futuristic cityscape at sunset"
image = pipe(prompt).images[0]
# Save the image
image.save("generated_image.png")
```

The following Python code helps us to generate music based on the user input (The code is generated using ChatGPT (Open Ai, 2025))

```
# Code generated with assistance from ChatGPT (OpenAI, 2025)
pip install torch torchaudio transformers
pip install git+https://github.com/facebookresearch/audiocraft#egg=audiocraft
from audiocraft.models import MusicGen
import torchaudio

# Load the MusicGen model (use 'small', 'medium', or 'melody')
model = MusicGen.get_pretrained('small')

# Input your prompt
prompt = input("Enter your music description (e.g., 'a chill lo-fi beat'): ")

# Generate music from the text
model.set_generation_params(duration=10)  # seconds
waveforms = model.generate([prompt])

# Save to file
torchaudio.save("generated_music.wav", waveforms[0].cpu(), sample_rate=32000)
print("Music generated and saved as 'generated_music.wav'")
```