Problem Statement

As the world moves towards more sustainable development, there is a growing need for energy-efficient architectural designs. Traditional methods of designing energy-efficient buildings are time-consuming and heavily reliant on human expertise. This project uses Generative Adversarial Networks (GANs) to generate synthetic architectural designs that simulate energy-efficient buildings. By training a GAN on existing architectural data, the model can generate novel, energy-efficient designs that could inspire architects and engineers in real-world projects.

Objective

- To simulate energy-efficient architecture designs using GANs.
- To train a GAN using existing energy-efficiency data to create new building designs.
- To apply machine learning (specifically GANs) to a real-world problem of energy-efficient design.

System Requirements

To Train the model in Local System

Hardware:

- GPU (Recommended for faster training, e.g., Nvidia CUDA-compatible GPUs)
- RAM: 8 GB or more
- Storage: At least 5 GB of free space for datasets and model storage

Software:

- Python 3.x
- TensorFlow 2.x
- Pandas, NumPy
- Matplotlib (Optional for visualization)
- Jupyter Notebook or any Python IDE
- GPU drivers and CUDA (if available)

It is recommended to use google colab, so there will not be any need of setting up the environment.

Dataset

The dataset used in this project is called energy_architecture_data.csv. This dataset contains features of energy-efficient architecture designs and their corresponding energy efficiency values.

Code Explanation

```
# Importing Required Libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
```

- Pandas is used for data manipulation.
- NumPy is used for numerical operations.
- TensorFlow and Keras are used for building and training the GAN model.

```
# GPU Configuration
physical_devices = tf.config.experimental.list_physical_devices('GPU')

if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
    print("Using GPU")
else:
    print("No GPU found, using CPU")
```

This part checks whether a GPU is available. If it is, it enables GPU acceleration for training; otherwise, it defaults to using the CPU.

```
# Loading the Dataset
data = pd.read_csv('energy_architecture_data.csv')
data.head()
```

The dataset is loaded using Pandas. data.head() will show the first few rows of the dataset.

Defining the GAN Architecture

```
# Dimensions
input_dim = 100  # Random noise vector input for the generator
output_dim = data.shape[1]  # Number of features in the data
```

```
# Generator Model
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(128, activation="relu", input_dim=input_dim),
        layers.Dense(256, activation="relu"),
        layers.Dense(512, activation="relu"),
        layers.Dense(output_dim, activation="tanh")  # Output layer
    ])
    return model
```

The Generator network creates synthetic samples from random noise. It consists of fully connected layers with increasing neurons and uses ReLU activation for hidden layers and Tanh activation for the output layer (to match the range of the data).

```
# Discriminator Model
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Dense(512, activation="relu", input_shape=(output_dim,)),
        layers.Dense(256, activation="relu"),
        layers.Dense(128, activation="relu"),
        layers.Dense(1, activation="sigmoid")  # Binary classification
    ])
    return model
```

The Discriminator network is a binary classifier that distinguishes between real and fake samples. It uses ReLU activation for hidden layers and Sigmoid activation for the output layer (to output probabilities).

```
# Instantiating and Compiling Models
generator = build_generator()
discriminator = build_discriminator()

discriminator.compile(optimizer=tf.keras.optimizers.Adam(0.0002, 0.5), loss="binary_crossent
```

- The generator and discriminator models are instantiated.
- The discriminator is compiled with the Adam optimizer and binary crossentropy loss, as it performs binary classification (real vs. fake).

```
# Building the GAN Model
discriminator.trainable = False  # Freeze the discriminator's weights when training the GAN
gan_input = layers.Input(shape=(input_dim,))
```

```
fake_data = generator(gan_input)
gan_output = discriminator(fake_data)
gan = tf.keras.Model(gan_input, gan_output)
gan.compile(optimizer=tf.keras.optimizers.Adam(0.0002, 0.5), loss="binary_crossentropy")
```

- The GAN model is constructed by connecting the generator and the discriminator.
- The discriminator's weights are frozen during GAN training (it is only updated when training the discriminator).

```
# Training the GAN Model
def train(data, epochs, batch_size):
    half_batch = int(batch_size / 2)

    for epoch in range(epochs):
        # --------------------- Train Discriminator ---------------------
        # Set discriminator trainable to True
        discriminator.trainable = True

        # Select a random half-batch of real samples
        idx = np.random.randint(0, data.shape[0], half_batch)
        real_data = data.iloc[idx].values

        # Generate a half-batch of fake samples
        noise = np.random.normal(0, 1, (half_batch, input_dim))
        fake_data = generator.predict(noise)

        # Train the discriminator on real and fake samples
        d_loss_real = discriminator.train_on_batch(real_data, np.ones((half_batch, 1)))
        d_loss_fake = discriminator.train_on_batch(fake_data, np.zeros((half_batch, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # --------------------- Train Generator ---------------------
        # Set discriminator trainable to False for GAN training
        discriminator.trainable = False

        # Generate a batch of noise
        noise = np.random.normal(0, 1, (batch_size, input_dim))

        # Train the generator
        g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))

        # Print progress
        if epoch % 100 == 0:
            print(f"epoch: {epoch+1}/{epochs} [D loss: {d_loss[0]:.4f}, acc.: {100*d_loss[1]
        else:
            print(f"epoch: {epoch+1}/{epochs} ...")
```

- This function trains the GAN for a set number of epochs.
- In each epoch, the discriminator is trained with both real and fake data, and the generator is trained to fool the discriminator into thinking fake data is real.

```
# Train the model
train(data, epochs=5000, batch_size=64)
```

```
# Saving the Model
generator.save('generator_model.h5')
```

After training, the generator model is saved to a file (generator_model.h5) for later use.

Generating Synthetic Data

```
# Function generate synthetic data
def generate_synthetic_data(generator, num_samples=1):
    noise_dim=100
    # Generate random noise
    noise = np.random.normal(0, 1, (num_samples, noise_dim))

    # Generate synthetic data using the generator
    synthetic_data = generator.predict(noise)
    return synthetic_data
```

This function generates synthetic architectural designs using the trained generator model. The generator is given random noise as input and produces synthetic data as output.

```
# Generating 10 synthetic samples
generate_synthetic_data(generator, num_samples=10)
```

```
# Loading the Saved Model
from tensorflow.keras.models import load_model
loaded_generator = load_model('generator_model.h5')
```