```python
import tensorflow as tf
from tensorflow.keras import layers, models, datasets
import numpy as np
import matplotlib.pyplot as plt
import os

CONFIG = {
    'dataset_choice': 'mnist',
    'epochs': 30,
    'batch_size': 128,
    'noise_dim': 100,
    'learning_rate': 0.0002,
    'save_interval': 5
}

os.makedirs('generated_samples', exist_ok=True)
os.makedirs('final_generated_images', exist_ok=True)

print(f"Configuration Loaded: {CONFIG}")

def load_data(choice):
    if choice == 'mnist':
        (x_train, y_train), (_, _) = datasets.mnist.load_data()
        print("Loaded MNIST dataset.")
    elif choice == 'fashion':
        (x_train, y_train), (_, _) = datasets.fashion_mnist.load_data()
        print("Loaded Fashion-MNIST dataset.")
    else:
        raise ValueError("Invalid dataset choice. Use 'mnist' or 'fashion'.")

    x_train = (x_train.astype('float32') - 127.5) / 127.5
    x_train = np.expand_dims(x_train, axis=3)

    return x_train, y_train

X_train, Y_train = load_data(CONFIG['dataset_choice'])

dataset = tf.data.Dataset.from_tensor_slices(X_train).shuffle(60000).batch(CONFIG['batch_size'])

def build_and_train_classifier(x_data, y_data):
    print("\n--- Preparing 'Pre-trained' Classifier for Final Evaluation ---")
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    model.fit(x_data, y_data, epochs=2, batch_size=64, verbose=1)
    print("Classifier ready.\n")
    return model

classifier_model = build_and_train_classifier(X_train, Y_train)

def build_generator(noise_dim):
    model = models.Sequential(name="Generator")
    model.add(layers.Dense(7 * 7 * 256, use_bias=False, input_shape=(noise_dim,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    return model

def build_discriminator():
    model = models.Sequential(name="Discriminator")
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))
    return model

generator = build_generator(CONFIG['noise_dim'])
discriminator = build_discriminator()

cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
generator_optimizer = tf.keras.optimizers.Adam(CONFIG['learning_rate'])
discriminator_optimizer = tf.keras.optimizers.Adam(CONFIG['learning_rate'])

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

def calculate_accuracy(real_output, fake_output):
    real_preds = tf.cast(real_output > 0, tf.float32)
    fake_preds = tf.cast(fake_output < 0, tf.float32)
    acc = (tf.reduce_mean(real_preds) + tf.reduce_mean(fake_preds)) / 2.0
    return acc * 100

@tf.function
def train_step(images):
    noise = tf.random.normal([CONFIG['batch_size'], CONFIG['noise_dim']])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)
        disc_acc = calculate_accuracy(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))

    return gen_loss, disc_loss, disc_acc

def save_plot(examples, epoch, n=5):
    examples = (examples + 1) / 2.0
    fig = plt.figure(figsize=(5, 5))
    for i in range(n * n):
        plt.subplot(n, n, i + 1)
        plt.axis('off')
        plt.imshow(examples[i, :, :, 0], cmap='gray')

    filename = f"generated_samples/epoch_{epoch:02d}.png"
    plt.savefig(filename)
    plt.close()

print("\nStarting GAN Training...")

seed = tf.random.normal([25, CONFIG['noise_dim']])

for epoch in range(1, CONFIG['epochs'] + 1):
    d_losses = []
    g_losses = []
    d_accs = []

    for image_batch in dataset:
        g_loss, d_loss, d_acc = train_step(image_batch)
        d_losses.append(d_loss)
        g_losses.append(g_loss)
        d_accs.append(d_acc)

    avg_d_loss = np.mean(d_losses)
    avg_g_loss = np.mean(g_losses)
    avg_d_acc = np.mean(d_accs)

    print(f"Epoch {epoch}/{CONFIG['epochs']} | D_loss: {avg_d_loss:.2f} | D_acc: {avg_d_acc:.2f}% | G_loss: {avg_g_loss:.2f}")

    if epoch % CONFIG['save_interval'] == 0:
        predictions = generator(seed, training=False)
        save_plot(predictions, epoch)
```

```python
    print("\nTraining Complete. Generating Final Evaluation...")

    final_noise = tf.random.normal([100, CONFIG['noise_dim']])
    final_images = generator(final_noise, training=False)

    fig = plt.figure(figsize=(10, 10))
    for i in range(100):
        plt.subplot(10, 10, i + 1)
        plt.axis('off')
        img = (final_images[i] + 1) / 2.0
        plt.imshow(img[:, :, 0], cmap='gray')
    plt.savefig('final_generated_images/final_grid_100.png')
    plt.close()
    print("Saved 100 generated images to 'final_generated_images/final_grid_100.png'")

    print("\nPredicting labels for generated images...")

    preds = classifier_model.predict(final_images, verbose=0)
    pred_labels = np.argmax(preds, axis=1)

    if CONFIG['dataset_choice'] == 'fashion':
        label_map = {0:'T-shirt/top', 1:'Trouser', 2:'Pullover', 3:'Dress', 4:'Coat',
                     5:'Sandal', 6:'Shirt', 7:'Sneaker', 8:'Bag', 9:'Ankle boot'}
        label_names = [label_map[l] for l in pred_labels]
    else:
        label_names = pred_labels

    unique, counts = np.unique(pred_labels, return_counts=True)
    print("\n--- Generated Image Label Distribution ---")
    for lbl, count in zip(unique, counts):
        name = label_map[lbl] if CONFIG['dataset_choice'] == 'fashion' else str(lbl)
        print(f"Label {name}: {count} images")

    print("\nProcess Finished.")
```

```
Configuration Loaded: {'dataset_choice': 'mnist', 'epochs': 30, 'batch_size': 128, 'noise_dim': 100, 'learning_rate': 0.0002, 'save_inter
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Loaded MNIST dataset.

--- Preparing 'Pre-trained' Classifier for Final Evaluation ---
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/2
938/938 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8906 - loss: 0.3672
Epoch 2/2
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9854 - loss: 0.0479
Classifier ready.

/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Starting GAN Training...
Epoch 1/30 | D_loss: 0.89 | D_acc: 79.58% | G_loss: 1.41
Epoch 2/30 | D_loss: 0.93 | D_acc: 78.36% | G_loss: 1.41
Epoch 3/30 | D_loss: 1.02 | D_acc: 75.75% | G_loss: 1.46
Epoch 4/30 | D_loss: 1.09 | D_acc: 72.18% | G_loss: 1.18
Epoch 5/30 | D_loss: 1.07 | D_acc: 73.83% | G_loss: 1.24
Epoch 6/30 | D_loss: 0.96 | D_acc: 77.55% | G_loss: 1.47
Epoch 7/30 | D_loss: 0.98 | D_acc: 76.73% | G_loss: 1.47
Epoch 8/30 | D_loss: 0.97 | D_acc: 76.90% | G_loss: 1.47
Epoch 9/30 | D_loss: 0.97 | D_acc: 76.89% | G_loss: 1.45
Epoch 10/30 | D_loss: 0.92 | D_acc: 78.43% | G_loss: 1.55
Epoch 11/30 | D_loss: 0.96 | D_acc: 76.94% | G_loss: 1.51
Epoch 12/30 | D_loss: 0.93 | D_acc: 78.34% | G_loss: 1.58
Epoch 13/30 | D_loss: 1.01 | D_acc: 75.78% | G_loss: 1.50
Epoch 14/30 | D_loss: 1.03 | D_acc: 74.72% | G_loss: 1.37
Epoch 15/30 | D_loss: 1.08 | D_acc: 72.74% | G_loss: 1.28
Epoch 16/30 | D_loss: 1.08 | D_acc: 72.68% | G_loss: 1.28
Epoch 17/30 | D_loss: 1.11 | D_acc: 72.06% | G_loss: 1.23
Epoch 18/30 | D_loss: 1.12 | D_acc: 71.12% | G_loss: 1.18
Epoch 19/30 | D_loss: 1.10 | D_acc: 72.24% | G_loss: 1.26
Epoch 20/30 | D_loss: 1.12 | D_acc: 71.63% | G_loss: 1.22
Epoch 21/30 | D_loss: 1.15 | D_acc: 70.18% | G_loss: 1.14
Epoch 22/30 | D_loss: 1.17 | D_acc: 69.14% | G_loss: 1.08
Epoch 23/30 | D_loss: 1.17 | D_acc: 69.15% | G_loss: 1.09
Epoch 24/30 | D_loss: 1.17 | D_acc: 69.35% | G_loss: 1.11
Epoch 25/30 | D_loss: 1.16 | D_acc: 69.78% | G_loss: 1.11
Epoch 26/30 | D_loss: 1.18 | D_acc: 68.70% | G_loss: 1.06
Epoch 27/30 | D_loss: 1.17 | D_acc: 69.40% | G_loss: 1.09
Epoch 28/30 | D_loss: 1.19 | D_acc: 68.19% | G_loss: 1.05
Epoch 29/30 | D_loss: 1.18 | D_acc: 68.60% | G_loss: 1.05
Epoch 30/30 | D_loss: 1.19 | D_acc: 68.41% | G_loss: 1.04

Training Complete. Generating Final Evaluation...
Saved 100 generated images to 'final_generated_images/final_grid_100.png'

Predicting labels for generated images...

--- Generated Image Label Distribution ---
Label 0: 12 images
Label 1: 14 images
```