```
import pennylane as qml
import tensorflow as tf
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Reshape, UpSampling2D
from tensorflow.keras import Sequential
import numpy as np
import matplotlib.pyplot as plt
from qiskit import transpile
from qiskit_ibm_runtime import QiskitRuntimeService, Session, Sampler
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from PIL import Image
import os
from transformers import AutoModelForImageGeneration, AutoTokenizer

IBM_API_KEY = 'Your API Key '
try:
    service = QiskitRuntimeService()
    print("IBM Quantum account loaded successfully.")
except Exception as e:
    print(f"Error loading IBM Quantum account: {e}")
backend_name = "ibmq_brisbane"
try:
    backend = service.backend(backend_name)
    print(f"Using backend: {backend_name}")
except Exception as e:
    print(f"Error accessing backend '{backend_name}': {e}")
    backend = None
optimization_level = 1
try:
    pass_manager = generate_preset_pass_manager(optimization_level=optimization_level, backend=backend)
        print(f"Pass manager with optimization level {optimization_level} created successfully.")
except Exception as e:
    print(f"Error creating pass manager: {e}")
    pass_manager = None

dev = qml.device("default.qubit", wires=4)
@qml.qnode(dev, interface="tf")
def quantum_circuit(latent_inputs):
    qml.AngleEmbedding(latent_inputs, wires=range(4))
    qml.RX(latent_inputs[0], wires=0)
    qml.RY(latent_inputs[1], wires=1)
    qml.RZ(latent_inputs[2], wires=2)
    qml.RX(latent_inputs[3], wires=3)
    return [qml.expval(qml.PauliZ(i)) for i in range(4)]
```

```
44.
45. def run_ibm_quantum_circuit(latent_inputs):
46.     if backend is None or pass_manager is None:
47.         print("Backend or Pass Manager is not properly initialized.")
48.         return np.zeros(64)
49.     try:
50.         qc = quantum_circuit(latent_inputs)
51.         transpiled_qc = pass_manager.run(qc)
52.         with Session(service=service, backend=backend) as session:
53.             sampler = Sampler(session=session)
54.             job = sampler.run(transpiled_qc)
55.             result = job.result()
56.             counts = result.get_counts()
57.         features = [counts.get(f"{i:07b}", 0) for i in range(64)]
58.         return features
59.     except Exception as e:
60.         print(f"Error in running quantum circuit: {e}")
61.         return np.zeros(64)
62.
63. def quantum_generator(latent_dim):
64.     model = Sequential([
65.         Dense(latent_dim, activation='relu', input_shape=(latent_dim,)),
66.                                                 tf.keras.layers.Lambda(lambda x:
    tf.stack([run_ibm_quantum_circuit(latent_inputs) for latent_inputs in x], axis=1)),
67.         Flatten(),
68.         Dense(65536, activation='relu'),
69.         Reshape((256, 256, 1)),
70.         Conv2D(1, kernel_size=(3, 3), padding="same", activation='sigmoid'),
71.         UpSampling2D(size=(10, 10))
72.     ])
73.     return model
74.
75. def prompt_to_features(prompt):
76.     prompt_hash = hash(prompt) % (2**8)
77.     features = [int(x) for x in bin(prompt_hash)[2:].zfill(8)][:4]
78.     return tf.convert_to_tensor(features, dtype=tf.float32)
79.
80. def generate_image_with_gan(prompt):
81.     model_name = "CompVis/taming-transformers"
82.     model = AutoModelForImageGeneration.from_pretrained(model_name)
83.     tokenizer = AutoTokenizer.from_pretrained(model_name)
84.     inputs = tokenizer(prompt, return_tensors="pt")
85.     output = model.generate(inputs.input_ids)
86.     image = output[0]
87.     image = image.permute(1, 2, 0)
88.     plt.imshow(image.cpu().numpy(), cmap='gray')
89.     plt.axis('off')
90.     plt.show()
```

```
    plt.savefig("generated_image_hf.png")
    return "generated_image_hf.png"

def generate_and_display_image(prompt):
    quantum_features = prompt_to_features(prompt)
    latent_dim = 4
    generator = quantum_generator(latent_dim)
    noise = tf.random.normal((1, latent_dim))
        generated_image = generator(tf.concat([noise, quantum_features[None, :]], axis=1))
        plt.imshow(generated_image[0, :, :, 0], cmap='gray')
        plt.axis('off')
        plt.show()
        image_path = "generated_image.png"
        plt.imsave(image_path, generated_image[0, :, :, 0], cmap='gray')
        print(f"Generated image saved to {image_path}.")
        gan_image_path = generate_image_with_gan(prompt)
        print(f"Hugging Face GAN generated image saved to {gan_image_path}")
        return gan_image_path

    if __name__ == "__main__":
        prompt = "A futuristic Poké Ball with quantum elements"
        generate_and_display_image(prompt)
```