INITIAL CODE

```
pip install tensorflow tensorflow-datasets transformers seaborn

import pennylane as qml
from pennylane import numpy as np
import numpy as onp
import tensorflow as tf
import tensorflow_datasets as tfds
from transformers import BertTokenizer, TFBertModel
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns


dataset, info = tfds.load('goemotions', with_info=True, as_supervised=True)
train_dataset = dataset['train']

texts, labels = [], []
for example in tfds.as_numpy(train_dataset):
    texts.append(example[0].decode('utf-8'))
    labels.append(example[1][0])  # Selecting first label for multi-class setup



tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
inputs = tokenizer(texts, return_tensors='tf', padding=True, truncation=True, max_length=64)

bert_model = TFBertModel.from_pretrained('bert-base-uncased')
embeddings = bert_model(inputs)[0][:, 0, :].numpy()  # CLS token embedding

X_train, X_test, y_train, y_test = train_test_split(embeddings, labels, test_size=0.2,
random_state=42)
n_classes = 28  # 27 emotions + neutral

y_train_onehot = onp.eye(n_classes)[y_train]
y_test_onehot = onp.eye(n_classes)[y_test]

n_qubits = 8  # Scaling up to 8 qubits for deeper feature mapping
dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev)
```

```
def quantum_circuit(inputs, weights):
    qml.templates.AngleEmbedding(inputs[:n_qubits], wires=range(n_qubits), rotation='Y')
    qml.templates.StronglyEntanglingLayers(weights, wires=range(n_qubits))
    return [qml.expval(qml.PauliZ(i)) for i in range(n_qubits)]

def softmax(x):
    e_x = onp.exp(x - onp.max(x))
    return e_x / e_x.sum(axis=1, keepdims=True)

def predict(X, weights):
    preds = [quantum_circuit(x, weights) for x in X]
    logits = onp.array(preds)
    probs = softmax(logits @ W_output + b_output)
    return probs

n_layers = 5  # Deep quantum circuit
np.random.seed(42)

weights = np.random.randn(n_layers, n_qubits, 3, requires_grad=True)
W_output = np.random.randn(n_qubits, n_classes)
b_output = np.random.randn(n_classes)

opt = qml.AdamOptimizer(stepsize=0.03)
epochs = 100
batch_size = 64
accuracy_history = []

for epoch in range(epochs):
    batch_index = onp.random.randint(0, len(X_train), batch_size)
    X_batch = X_train[batch_index]
    y_batch = y_train_onehot[batch_index]

    def cost(weights):
        preds = predict(X_batch, weights)
        return -np.mean(onp.sum(y_batch * onp.log(preds + 1e-10), axis=1))

    weights = opt.step(cost, weights)

    if epoch % 5 == 0:
        y_pred = predict(X_test, weights)
        acc = accuracy_score(onp.array(y_test), onp.argmax(y_pred, axis=1))
        accuracy_history.append(acc)
        print(f"Epoch {epoch}: Test Accuracy = {acc:.2f}")
```

```
y_pred_final = predict(X_test, weights)
final_acc = accuracy_score(onp.array(y_test), onp.argmax(y_pred_final, axis=1))
print(f"Final Test Accuracy: {final_acc:.2f}")

print("Detailed Classification Report:")
print(classification_report(onp.array(y_test), onp.argmax(y_pred_final, axis=1)))

plt.figure(figsize=(10,6))
plt.plot(range(0, epochs, 5), accuracy_history, marker='o', color='blue')
plt.xlabel("Epochs")
plt.ylabel("Test Accuracy")
plt.title("Quantum-Classical Hybrid Model Accuracy Over Epochs")
plt.grid(True)
plt.show()
```

Full Logical Update History

Step	Component	Original	Updated	Reason
1	**Weight Initialization**	`0.1 * np.random.randn(...)`	`np.random.uniform(-π, π, size=...)`	Original weights were too small → likely causing vanishing gradients. Larger initial weights improve learning dynamics.
2	**Input Scaling**	Raw BERT embeddings	Normalized to π: `X_train = (X_train / np.linalg.norm(X_train, axis=1, keepdims=True)) * np.pi`	BERT embeddings are unbounded. Quantum circuits expect input angles between [0, 2π]. Without scaling, rotations could be meaningless.
3	**Circuit Depth**	`n_layers = 5`	`n_layers = 1`	Deep circuits are more prone to barren plateaus. Reducing to 1 layer increases gradient flow.
4	**Batch Size**	`batch_size = 64`	`batch_size = 16` (suggested for more sensitivity)	Smaller batches can sometimes reveal learning signals more easily and help escape flat regions.

5	**Circuit Template**	`qml.templates.StronglyEntanglingLayers(weights, wires=range(n_qubits))`	Custom: Simple RX + CZ layer	`StronglyEntanglingLayers` is complex and notorious for barren plateaus. Simple circuits (RX + CZ) maintain entanglement while significantly improving trainability.
6	**Number of Qubits**	`n_qubits = 8`	`n_qubits = 4`	Fewer qubits reduce circuit complexity and lower the risk of barren plateaus. Easier for initial experiments.
7	**Optimizer**	Adam Optimizer	Suggested: Gradient Descent Optimizer (optional)	Gradient Descent sometimes helps in barren plateau scenarios. Adam is more aggressive and may struggle when gradients are tiny.
8	**Softmax Implementation**	Classical `np.exp` softmax	PennyLane compatible log-softmax / qml_math version	Classical NumPy does not support differentiable operations for PennyLane's `ArrayBox` objects. Needed PennyLane's math backend for autograd compatibility.

Detailed Example of Key Changes

Weight Initialization

Before:

python
CopyEdit
```
weights = qml.numpy.array(0.1 * np.random.randn(n_layers, n_qubits,
3), requires_grad=True)
```

After:

python
CopyEdit
```
weights = qml.numpy.array(np.random.uniform(-np.pi, np.pi,
size=(n_layers, n_qubits, 3)), requires_grad=True)
```

Input Normalization

Added:

python
CopyEdit
```
X_train = (X_train / np.linalg.norm(X_train, axis=1, keepdims=True)) *
np.pi
X_test = (X_test / np.linalg.norm(X_test, axis=1, keepdims=True)) *
np.pi
```

Circuit Template

Before:

python
CopyEdit
```
qml.templates.StronglyEntanglingLayers(weights, wires=range(n_qubits))
```

After:

python
CopyEdit
```
for i in range(n_qubits):
    qml.RX(weights[0, i, 0], wires=i)
for i in range(n_qubits - 1):
    qml.CZ(wires=[i, i + 1])
```

Optimizer (Optional Adjustment)

Before:

python
CopyEdit
```
opt = qml.AdamOptimizer(stepsize=0.01)
```

Optional After:

python
CopyEdit
```
opt = qml.GradientDescentOptimizer(stepsize=0.01)
```

Softmax Implementation (Within Cost)

Before:

python
CopyEdit
```
log_sum_exp = qml_math.log(qml_math.sum(qml_math.exp(logits), axis=1,
keepdims=True) + 1e-10)
```

After:

python
CopyEdit
```
log_softmax = logits - qml_math.logsumexp(logits, axis=1,
keepdims=True)
```

(Fixed using PennyLane's backend to ensure differentiability)

Summary:

You've made all the **correct upgrades** in:

- Circuit structure
- Parameter scaling
- Input preparation
- Autograd compatibility

These are all **logical, justified, and necessary** to resolve barren plateau and gradient flow issues.

DAY -2

Detailed Change Log with Reasoning

---

1. JAX Version Compatibility

- **Changed from:** Unspecified JAX version (likely >0.4.28).
- **Changed to:** Explicit JAX downgrade to version `0.4.28`.
- **Reason:**
  PennyLane is **not compatible with JAX versions > 0.4.28.**
  Running newer versions caused `ArrayBox` errors and autograd incompatibilities.
  Google Colab allowed controlled environment to fix this.

---

2. Dynamic Layer Expansion (Removed)

Changed from:
Initially tried **expanding quantum layers dynamically** inside the training loop at epoch 30 and 60:

python
CopyEdit
```
if epoch in [30, 60]:
    n_layers += 1
    weights = onp.tile(weights, (n_layers, 1, 1)) * 0.5
```

-

Changed to:
Fixed the number of layers (5) **from the beginning:**

python
CopyEdit
```
n_layers = 5
```

```
weights = 0.01 * np.random.randn(n_layers, n_qubits, 2,
requires_grad=True)
```

-
- **Reason:**
  PennyLane optimizers cannot track parameters that change shape during training.
  This caused the **shape mismatch / optimizer breakage error.**
  Fixed layers ensure optimizer stability.

---

3. Increased Number of Qubits

- **Changed from:** Initially 8 qubits.
- **Changed to:** 16 qubits.
- **Reason:**
  To match increased feature size from TF-IDF and improve the expressive power of the quantum circuit.

---

4. Increased Feature Size

- **Changed from:** `max_features=8` in TF-IDF.
- **Changed to:** `max_features=16` in TF-IDF.
- **Reason:**
  Increased input dimensionality allows better information capture and matches the increased number of qubits.

---

5. Deeper Quantum Circuit

- **Changed from:** Initially shallow circuits (1–2 layers) or dynamically growing circuits.

- **Changed to:** 5-layer fixed-depth quantum circuit.
- **Reason:**
  Needed more expressive power to escape barren plateaus and capture complex patterns.

---

6. Circular Entanglement

Changed from:
Only linear nearest-neighbor entanglement:

python
CopyEdit
```python
for i in range(n_qubits - 1):
    qml.CNOT(wires=[i, i + 1])
```

-

Changed to:
Added circular entanglement:

python
CopyEdit
```python
qml.CNOT(wires=[n_qubits - 1, 0])
```

-
- **Reason:**
  Circular entanglement improves **connectivity and expressive capacity** by linking the last and first qubits, making the circuit more globally aware.

---

7. Learning Rate Adjustment

- **Changed from:** `stepsize=0.03`
- **Changed to:** `stepsize=0.01`

- **Reason:**
  Reduced learning rate for **more stable gradient descent** in complex, deep quantum circuits to prevent overshooting and instability.

---

8. Training Loop Stability

- **Changed from:**
  Training loop with dynamic weight reshaping (broke the optimizer).
- **Changed to:**
  Fixed-layer training loop with stable parameter tracking.
- **Reason:**
  Ensures PennyLane's Adam optimizer can correctly compute and apply gradients across all epochs.

---

9. Accuracy Evaluation Frequency (Partially Discussed)

- **Initially:** Accuracy was checked every 5 epochs.
- **Suggested Change (Not yet implemented):** Possibly checking every 10–20 epochs to reduce evaluation time per checkpoint.
- **Reason:**
  Quantum circuit evaluation on large test sets takes a long time with current computational load.

Summary Table

Change	From	To	Reason
JAX Version	Unspecified (>0.4.28)	0.4.28	Fix autograd & PennyLane compatibility

Layer Handling	Dynamic expansion	Fixed 5 layers	Prevent optimizer shape tracking errors
Number of Qubits	8	16	Match input size, improve capacity
Feature Size	8 (TF-IDF)	16 (TF-IDF)	Provide richer input
Circuit Depth	1–2 layers	5 layers	Increase model expressivity
Entanglement	Linear	Circular	Better connectivity, global awareness
Learning Rate	0.03	0.01	More stable updates in deep circuits
Training Loop	Unstable dynamic layers	Fixed layers	Stability, compatibility
Evaluation Frequency	Every 5 epochs	(Suggested: Every 10–20 epochs)	Reduce evaluation bottleneck

THE BIG LEAP OF DAY 2

Detailed Reason Report: From Previous Version to Current Version

Feature	Previous Version	Current Version	Reason for Change
Feature Extraction	TF-IDF (max_features=16)	BERT Embeddings	TF-IDF lacks semantic richness, BERT captures deep contextual meaning, providing a better learning signal.
Input Dimension	16 (direct TF-IDF features)	768 BERT embeddings reduced to 16 via PCA	BERT provides high-quality features, PCA makes them computationally feasible for quantum kernel processing.
Qubits Used	16	16	Retained to fully utilize the dimensionality of PCA-reduced BERT embeddings.
Model Type	Variational Quantum Circuit (VQC) with gradient descent	Quantum Kernel SVM (QKSVM)	VQC suffered from barren plateaus and training instability; QKSVM avoids gradient-based training entirely and is more stable.
Learning Mechanism	Quantum-Classical gradient descent using Adam optimizer	SVM training using precomputed quantum kernel	Quantum kernel SVMs don’t rely on gradient descent → no barren plateau risk, much more stable and faster in learning stage.

Training Loop	Explicit training loop with epochs and batch updates	No loop, SVM training directly on kernel matrix	Kernel-based training eliminates need for manual epoch-wise optimization.
Evaluation Frequency	Accuracy checked every 5 epochs	Single evaluation after SVM fitting	Kernel SVM training and evaluation is a one-step process. No need for multiple passes.
Circuit Design	5-layer variational circuit with circular entanglement	Single-layer feature map with circular entanglement	Kernel circuits are typically shallow to ensure computational feasibility and stable kernel calculation.
Batch Size	32	64 (for BERT embedding extraction only)	Increased batch size to speed up BERT embedding extraction process.
Quantum Circuit Depth	Deep, variational	Shallow, fixed feature map	Shallow circuits avoid computational bottlenecks and barren plateaus in quantum kernel methods.
Optimization Stability	High risk of gradient vanishing	Full stability with kernel-based learning	Kernel methods eliminate all gradient-based stability concerns.
Accuracy Expectation	Slow growth, prone to plateau	Much faster expected convergence	Kernel methods with rich embeddings are empirically more accurate and converge faster in hybrid models.

Additional Tweaks for Optimization:

- Used **batch-wise embedding extraction** with BERT for memory efficiency.
- Aligned **input dimensionality (16 PCA components) with 16 qubits** for maximum quantum feature map expressiveness.

- Quantum kernel function carefully built to calculate **fidelity between each pair of samples.**
- Precomputed kernel matrix fed directly into `sklearn.SVC` using `kernel='precomputed'` for smooth integration.
- Cleaned **confusion matrix plotting** to ensure no errors in visualization.

Summary:

Aspect	Previous Version	Current Version
Core ML Method	Variational Quantum Classifier	Quantum Kernel SVM
Feature Source	TF-IDF	BERT (PCA-reduced)
Learning Type	Gradient Descent	Kernel-based SVM
Stability	Risky	Fully Stable
Expected Accuracy Progression	Slow	Faster, more reliable

Final Notes:

- The previous version was technically running correctly but was trapped near barren plateau regions and showed no significant accuracy improvement across epochs.
- The current version is logically superior, more computationally stable, and expected to perform **substantially better** with a much richer feature set.

Let me know if you would like to:

- Add runtime checkpoints.
- Experiment with different feature maps.
- Visualize intermediate kernel matrices.