In [9]:
```python
# Classical Thematic Correlator Experiment
# Objective: Classify documents describing a thematic link using engineered features.
#

import os
import time
import numpy as np
from dotenv import load_dotenv

# --- Imports for this experiment ---
from pymongo import MongoClient
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report, accuracy_score

# --- CONFIGURATION ---
# Define the core concepts for our experiment
IMMUNE_KEYWORDS = ["immune", "cytokines", "t-cells", "inflammation", "inflammatory"]
NEURO_KEYWORDS = ["neuro", "alzheimer's", "parkinson's", "neurons", "neurodegenerative"]
LINKING_WORDS = ["factor", "driven", "progression", "connect", "link", "cause", "pathway", "role"]
RANDOM_SEED = 1337

# ==============================================================================
#           PART 1: DYNAMIC DATASET CREATION FROM MONGODB
# ==============================================================================

def discover_training_candidates(collection, limit_per_category=50) -> tuple[np.ndarray, np.ndarray]:
    """Queries MongoDB to find documents for our thematic link experiment."""
    print("\n--- Discovering training candidates from MongoDB ---")

    # Query for "Thematic Link" documents (Positive Examples)
    query_link = {
        "$and": [
            {"content": {"$regex": "|".join(IMMUNE_KEYWORDS), "$options": "i"}},
            {"content": {"$regex": "|".join(NEURO_KEYWORDS), "$options": "i"}},
            {"content": {"$regex": "|".join(LINKING_WORDS), "$options": "i"}}
        ]
    }
    link_docs = list(collection.find(query_link).limit(limit_per_category))

    # Query for "Simple Co-occurrence" documents (Hard Negative Examples)
```

```
    query_co_occurrence = {
        "$and": [
            {"content": {"$regex": "|".join(IMMUNE_KEYWORDS), "$options": "i"}},
            {"content": {"$regex": "|".join(NEURO_KEYWORDS), "$options": "i"}},
            {"content": {"$not": {"$regex": "|".join(LINKING_WORDS), "$options": "i"}}}
        ]
    }
    co_occurrence_docs = list(collection.find(query_co_occurrence).limit(limit_per_category))

    # Create the final dataset and labels
    documents = link_docs + co_occurrence_docs
    labels = np.array([1] * len(link_docs) + [0] * len(co_occurrence_docs))

    print(f"Discovered {len(documents)} total candidates: {len(link_docs)} 'Link' (Label 1) and {len(co_occurrence_docs)} '(
    return np.array(documents), labels

# ==============================================================================
#           PART 2: THE CLASSICAL LINK CLASSIFIER
# ==============================================================================

class ClassicalLinkClassifier:
    def __init__(self, random_seed=RANDOM_SEED):
        self.model = SGDClassifier(loss='log_loss', random_state=random_seed)

    def _extract_link_features(self, document: dict) -> np.ndarray:
        """Extracts manually engineered features to detect a 'link'."""
        doc_text = (document.get("title", "") + " " + document.get("content", "")).lower()
        doc_words = doc_text.split()

        # Feature 1: Presence of linking words
        has_linking_words = 1.0 if any(word in doc_words for word in LINKING_WORDS) else 0.0

        # Feature 2 & 3: Presence of concept keywords
        immune_present = 1.0 if any(word in doc_words for word in IMMUNE_KEYWORDS) else 0.0
        neuro_present = 1.0 if any(word in doc_words for word in NEURO_KEYWORDS) else 0.0

        return np.array([has_linking_words, immune_present, neuro_present])

    def train(self, X_train_docs, y_train):
        print("\n--- Training Classical Model with Engineered Features ---")
        X_train_features = np.array([self._extract_link_features(doc) for doc in X_train_docs])
        self.model.fit(X_train_features, y_train)
        print("Training complete.")
```

```
    def predict(self, X_test_docs):
        print("\n--- Evaluating Classical Model ---")
        X_test_features = np.array([self._extract_link_features(doc) for doc in X_test_docs])
        return self.model.predict(X_test_features)

# ==============================================================================
#           PART 3: MAIN EXPERIMENT EXECUTION
# ==============================================================================

if __name__ == "__main__":
    print(f"Starting Classical Thematic Correlator Experiment... (Timestamp: {time.time()}, Location: Bengaluru, India)")
    load_dotenv()
    mongo_uri = os.getenv("MONGO_URI")
    db_name = os.getenv("MONGO_DB")
    collection_name = os.getenv("MONGO_COLLECTION")
    if not all([mongo_uri, db_name, collection_name]):
        raise ValueError("MongoDB credentials not found in .env file.")

    mongo_client = MongoClient(mongo_uri)
    db = mongo_client[db_name]
    collection = db[collection_name]
    print("MongoDB connection successful.")

    # 1. Create dataset from the database
    documents, labels = discover_training_candidates(collection, limit_per_category=50)

    if len(np.unique(labels)) < 2:
        print("Could not find enough documents for both classes. Experiment requires 'Link' and 'Co-occurrence' docs. Abort
    else:
        # 2. Split data for training and testing
        X_train_docs, X_test_docs, y_train, y_test = train_test_split(
            documents, labels, test_size=0.3, random_state=RANDOM_SEED, stratify=labels
        )
        print(f"\nCreated dataset with {len(y_train)} training samples and {len(y_test)} testing samples.")

        # 3. Run Classical Experiment
        classical_model = ClassicalLinkClassifier()
        classical_model.train(X_train_docs, y_train)
        classical_preds = classical_model.predict(X_test_docs)
        classical_accuracy = accuracy_score(y_test, classical_preds)

        print("\n--- FINAL CLASSICAL MODEL RESULTS ---")
```

```
        print(f"Accuracy (Engineered Features): {classical_accuracy:.2%}")
        print("\nClassification Report:")
        print(classification_report(y_test, classical_preds))

    mongo_client.close()
    print("\nMongoDB connection closed. Experiment finished.")
```

```
Starting Classical Thematic Correlator Experiment... (Timestamp: 1755755813.202131, Location: Bengaluru, India)
MongoDB connection successful.

--- Discovering training candidates from MongoDB ---
Discovered 77 total candidates: 50 'Link' (Label 1) and 27 'Co-occurrence' (Label 0).

Created dataset with 53 training samples and 24 testing samples.

--- Training Classical Model with Engineered Features ---
Training complete.

--- Evaluating Classical Model ---

--- FINAL CLASSICAL MODEL RESULTS ---
Accuracy (Engineered Features): 70.83%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.12      0.22         8
           1       0.70      1.00      0.82        16

    accuracy                           0.71        24
   macro avg       0.85      0.56      0.52        24
weighted avg       0.80      0.71      0.62        24


MongoDB connection closed. Experiment finished.
```

In [21]:

```
#
# Quantum Thematic Correlator Experiment
# Final Authoritative Version with Error Mitigation and Hardware Optimization
#

import os
import time
import numpy as np
```

```python
from dotenv import load_dotenv

# --- Data and NLP Imports ---
from pymongo import MongoClient
from sentence_transformers import SentenceTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics.pairwise import cosine_similarity

# --- Modern Qiskit Imports for the Open Plan ---
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2 as Sampler
# --- CORRECTED IMPORT: Import the specific options class for clarity and validation ---
from qiskit_ibm_runtime.options import SamplerOptions
from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes
from qiskit.circuit import QuantumCircuit
from qiskit_algorithms.optimizers import COBYLA
from qiskit_algorithms.utils import algorithm_globals
from qiskit.compiler import transpile

# --- CONFIGURATION ---
algorithm_globals.random_seed = 1337
IMMUNE_KEYWORDS = ["immune", "cytokines", "t-cells", "inflammation", "inflammatory"]
NEURO_KEYWORDS = ["neuro", "alzheimer's", "parkinson's", "neurons", "neurodegenerative"]
LINKING_WORDS = ["factor", "driven", "progression", "connect", "link", "cause", "pathway", "role"]

def discover_training_candidates(collection, limit_per_category=50) -> tuple[np.ndarray, np.ndarray]:
    """Queries MongoDB to find documents for our thematic link experiment."""
    print("\n--- Discovering training candidates from MongoDB ---")
    query_link = {
        "$and": [
            {"content": {"$regex": "|".join(IMMUNE_KEYWORDS), "$options": "i"}},
            {"content": {"$regex": "|".join(NEURO_KEYWORDS), "$options": "i"}},
            {"content": {"$regex": "|".join(LINKING_WORDS), "$options": "i"}}
        ]
    }
    link_docs = list(collection.find(query_link).limit(limit_per_category))

    query_co_occurrence = {
        "$and": [
            {"content": {"$regex": "|".join(IMMUNE_KEYWORDS), "$options": "i"}},
            {"content": {"$regex": "|".join(NEURO_KEYWORDS), "$options": "i"}},
            {"content": {"$not": {"$regex": "|".join(LINKING_WORDS), "$options": "i"}}}
        ]
```

```
    }
    co_occurrence_docs = list(collection.find(query_co_occurrence).limit(limit_per_category))

    documents = link_docs + co_occurrence_docs
    labels = np.array([1] * len(link_docs) + [0] * len(co_occurrence_docs))

    print(f"Discovered {len(documents)} total candidates: {len(link_docs)} 'Link' (Label 1) and {len(co_occurrence_docs)} '(
    return np.array(documents), labels


class QuantumLinkClassifier:
    """
    A quantum classifier refactored to use the manual, session-less pattern
    required for the IBM Quantum Open Plan. Includes hardware-aware optimizations.
    """
    def __init__(self, service, backend_name="ibm_brisbane"):
        print(f"\n--- Initializing Quantum Link Classifier for '{backend_name}' ---")

        # --- 1. Initialize Primitives and Backend ---
        self.backend_name = backend_name
        self.shots = 4096
        backend_object = service.backend(self.backend_name)

        # --- CORRECTED: Initialize the SamplerOptions class ---
        # This provides a structured way to set options and avoids validation errors.
        options = SamplerOptions()

        # --- NOTE: SamplerV2 does NOT support the 'resilience_level' option. ---
        # This feature is specific to EstimatorV2 for mitigating errors in expectation values.
        # When using SamplerV2, the backend may still apply some default level of readout
        # error correction, but the advanced mitigation techniques controlled by
        # resilience_level are not available. The code block attempting to set this
        # has been removed to fix the error.
        if not backend_object.configuration().simulator:
            print("Real hardware detected. The backend will apply its default error correction.")

        print("Initializing Sampler with V2 options...")
        # Pass the structured options object to the Sampler
        self.sampler = Sampler(mode=backend_object, options=options)
        print("Sampler initialized successfully.")

        # --- 2. Create and Transpile the Quantum Circuit ---
        self.feature_dim = 2
```

```
        feature_map = ZZFeatureMap(feature_dimension=self.feature_dim, reps=2)

        # --- STRATEGY: Simplify the Ansatz for Noise Resilience ---
        print("Using a noise-resilient ansatz with reps=2.")
        self.ansatz = RealAmplitudes(num_qubits=self.feature_dim, reps=2)

        pqc = QuantumCircuit(self.feature_dim)
        pqc.compose(feature_map, inplace=True)
        pqc.compose(self.ansatz, inplace=True)
        pqc.measure_all(inplace=True)

        print("Abstract PQC created. Transpiling for hardware compatibility...")
        # --- STRATEGY: Use optimization_level=1 for hardware to prioritize noise resilience over aggressive gate reduction
        self.isa_pqc = transpile(pqc, backend=backend_object, optimization_level=1)
        print("Transpilation complete.")

        # --- 3. Classical Components ---
        self.embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
        self.concept_A_embedding = self.embedding_model.encode(["immune system response inflammation"])
        self.concept_B_embedding = self.embedding_model.encode(["neurodegenerative disease alzheimer's parkinson's"])
        self.optimal_weights = None

    def _extract_semantic_features(self, document: dict) -> np.ndarray:
        doc_text = document.get("title", "") + " " + document.get("content", "")
        if not doc_text.strip(): return np.array([0.0, 0.0])
        doc_embedding = self.embedding_model.encode([doc_text])
        sim_A = cosine_similarity(doc_embedding, self.concept_A_embedding)[0][0]
        sim_B = cosine_similarity(doc_embedding, self.concept_B_embedding)[0][0]
        return np.array([sim_A, sim_B])

    def train(self, X_train_docs, y_train, maxiter=5):
        print(f"\n--- Starting Manual Training ({maxiter} optimizer iterations) ---")
        X_train_features = np.array([self._extract_semantic_features(doc) for doc in X_train_docs])
        optimizer = COBYLA(maxiter=maxiter)

        iteration_count = 0
        def objective_function(weights):
            nonlocal iteration_count
            iteration_count += 1
            print(f"\n--- Optimizer Iteration: {iteration_count}/{maxiter} ---")

            # A PUB (Primitive Unified Bloc) is a tuple of (circuit, parameter_values)
            pubs = [(self.isa_pqc, np.concatenate((x_i, weights))) for x_i in X_train_features]
```

```
            print(f"Submitting job with {len(pubs)} PUBs...")
            # For SamplerV2, shots is an argument to the run() method.
            job = self.sampler.run(pubs, shots=self.shots)
            print(f"Job submitted with ID: {job.job_id()}. Waiting for results...")
            result = job.result()
            print("Results received.")

            total_loss = 0
            for i, y_true in enumerate(y_train):
                pub_result = result[i]
                # Access measurement outcomes via the 'meas' data field
                outcomes = pub_result.data.meas.array
                # Calculate probability of '1' state (assuming standard Z measurement on the first qubit)
                prob_1 = np.mean(outcomes % 2)
                total_loss += (prob_1 - y_true)**2

            avg_loss = total_loss / len(y_train)
            print(f"  Avg. Loss for Iteration {iteration_count}: {avg_loss:.4f}")
            return avg_loss

        initial_weights = np.random.uniform(0, 2 * np.pi, self.ansatz.num_parameters)
        opt_result = optimizer.minimize(objective_function, initial_weights)
        self.optimal_weights = opt_result.x
        print("\n--- Training Complete ---")

    def predict(self, X_test_docs):
        print("\n--- Evaluating Quantum Model ---")
        if self.optimal_weights is None:
            raise RuntimeError("Model must be trained first.")

        X_test_features = np.array([self._extract_semantic_features(doc) for doc in X_test_docs])
        pubs = [(self.isa_pqc, np.concatenate((x_i, self.optimal_weights))) for x_i in X_test_features]

        print(f"Submitting prediction job with {len(pubs)} PUBs...")
        # For SamplerV2, shots is an argument to the run() method.
        job = self.sampler.run(pubs, shots=self.shots)
        print(f"Job submitted with ID: {job.job_id()}. Waiting for results...")
        result = job.result()
        print("Prediction results received.")

        predictions = []
        for pub_result in result:
```

```
            outcomes = pub_result.data.meas.array
            prob_1 = np.mean(outcomes % 2)
            predictions.append(1 if prob_1 > 0.5 else 0)

        return np.array(predictions)


if __name__ == "__main__":
    print(f"Starting Quantum Thematic Correlator Experiment... (Timestamp: {time.time()}, Location: Bengaluru, India)")
    load_dotenv()

    # --- EXPERIMENT CONFIGURATION ---
    # 1. To get a NOISELESS BASELINE, use "ibm_qasm_simulator" and a high maxiter.
    # 2. To run on HARDWARE, use a real backend like "ibm_brisbane" and a low maxiter.
    BACKEND_NAME = "ibm_brisbane"

    if "simulator" in BACKEND_NAME:
        MAX_ITERATIONS = 50 # More iterations for the fast, ideal simulator
    else:
        MAX_ITERATIONS = 5  # Fewer iterations for the slower hardware queue

    # --- Service Initialization ---
    # The 'instance' is a specific project group. 'ibm-q/open/main' is the standard for the open plan.
    # NOTE: The token below is a placeholder and has been kept as-is from the original script.
    # In a real scenario, this should be loaded securely, e.g., from environment variables.
    instance_name = "ibm_quantum" # A common instance for open plan users
    service = QiskitRuntimeService(
        channel='ibm_quantum_platform',
        token=YOUR_IBM_KEY,
        instance=instance_name
    )
    # --- CORRECTED: Use the variable holding the instance name for the print statement ---
    print(f"Service initialized for instance '{instance_name}'.")

    # --- MongoDB Connection ---
    # Make sure your .env file has MONGO_URI, MONGO_DB, and MONGO_COLLECTION set
    mongo_client = MongoClient(os.getenv("MONGO_URI"))
    db = mongo_client[os.getenv("MONGO_DB")]
    collection = db[os.getenv("MONGO_COLLECTION")]
    print("MongoDB connection successful.")

    documents, labels = discover_training_candidates(collection, limit_per_category=50)
```

```
    if len(np.unique(labels)) < 2:
        print("Could not find enough docs for both classes. Aborting.")
    else:
        X_train_docs, X_test_docs, y_train, y_test = train_test_split(
            documents, labels, test_size=0.3, random_state=1337, stratify=labels
        )
        print(f"\nCreated dataset with {len(y_train)} training samples and {len(y_test)} testing samples.")

        # --- Run Quantum Experiment ---
        quantum_model = QuantumLinkClassifier(service=service, backend_name=BACKEND_NAME)
        quantum_model.train(X_train_docs, y_train, maxiter=MAX_ITERATIONS)
        quantum_preds = quantum_model.predict(X_test_docs)

        print("\n--- FINAL QUANTUM MODEL RESULTS ---")
        print(f"Backend: {BACKEND_NAME}")
        print(f"Accuracy (Simple Semantic Features): {accuracy_score(y_test, quantum_preds):.2%}")
        print("\nClassification Report:")
        print(classification_report(y_test, quantum_preds))

    mongo_client.close()
    print("\nMongoDB connection closed. Experiment finished.")
```

```
Starting Quantum Thematic Correlator Experiment... (Timestamp: 1755761709.2004569, Location: Bengaluru, India)
Service initialized for instance 'ibm_quantum'.
MongoDB connection successful.

--- Discovering training candidates from MongoDB ---
Discovered 77 total candidates: 50 'Link' (Label 1) and 27 'Co-occurrence' (Label 0).

Created dataset with 53 training samples and 24 testing samples.

--- Initializing Quantum Link Classifier for 'ibm_brisbane' ---
Real hardware detected. The backend will apply its default error correction.
Initializing Sampler with V2 options...
Sampler initialized successfully.
Using a noise-resilient ansatz with reps=2.
Abstract PQC created. Transpiling for hardware compatibility...
Transpilation complete.

--- Starting Manual Training (5 optimizer iterations) ---

--- Optimizer Iteration: 1/5 ---
Submitting job with 53 PUBs...
Job submitted with ID: d2jcoffa6cjs73f8ugk0. Waiting for results...
Results received.
  Avg. Loss for Iteration 1: 0.3290

--- Optimizer Iteration: 2/5 ---
Submitting job with 53 PUBs...
Job submitted with ID: d2jcp0sg59ks73c3k510. Waiting for results...
Results received.
  Avg. Loss for Iteration 2: 0.3288

--- Optimizer Iteration: 3/5 ---
Submitting job with 53 PUBs...
Job submitted with ID: d2jcpi8hsgmc73b33aqg. Waiting for results...
Results received.
  Avg. Loss for Iteration 3: 0.2893

--- Optimizer Iteration: 4/5 ---
Submitting job with 53 PUBs...
Job submitted with ID: d2jcq3uhb60s73cruoeg. Waiting for results...
Results received.
  Avg. Loss for Iteration 4: 0.3684
```

```
--- Optimizer Iteration: 5/5 ---
Submitting job with 53 PUBs...
Job submitted with ID: d2jcqlghsgmc73b33bq0. Waiting for results...
Results received.
  Avg. Loss for Iteration 5: 0.2824

--- Training Complete ---

--- Evaluating Quantum Model ---
Submitting prediction job with 24 PUBs...
Job submitted with ID: d2jcr8kg59ks73c3k72g. Waiting for results...
Prediction results received.

--- FINAL QUANTUM MODEL RESULTS ---
Backend: ibm_brisbane
Accuracy (Simple Semantic Features): 62.50%

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         8
           1       0.65      0.94      0.77        16

    accuracy                           0.62        24
   macro avg       0.33      0.47      0.38        24
weighted avg       0.43      0.62      0.51        24


MongoDB connection closed. Experiment finished.
```

In [23]:
```
"""Experiment Report: A Comparison of Classical and Quantum Thematic Correlators
Date: Thursday, August 21, 2025
Location: Bengaluru, Karnataka, India
Author: Anirudh R
Project Status: Complete

Executive Summary
This report details the results of a head-to-head experiment comparing a classical machine learning
model with engineered features against a Variational Quantum Classifier (VQC) with simple semantic features.
Both models were tasked with identifying documents that describe a thematic link between two concepts
("Immune System" and "Neurodegenerative Disease"). The dataset was dynamically generated from a MongoDB
database, resulting in an imbalanced set of 77 documents.
```

The results show that the classical model achieved a significantly higher accuracy (70.83%) compared to the quantum model (62.50%). Analysis of the classification reports reveals that both models struggled with the imbalanced dataset, but the classical model learned a useful, albeit biased, decision boundary. The quantum model, running on the ibm_brisbane hardware, failed to learn a meaningful pattern and resorted to predicting the majority class. The primary conclusion is that for this task, the manually engineered features provided a much stronger signal than the quantum feature map was able to extract from simple semantic inputs.

1. Experiment Objective
The goal was to test if a quantum re-ranker, using simple semantic features, could outperform a classical one that required complex, manual feature engineering. The task was to classify documents as either describing a thematic link (Label 1) or merely containing a co-occurrence of keywords (Label 0).

2. Methodology
Dataset Discovery: A dataset was dynamically created by querying a MongoDB database. This process discovered 77 total candidate documents.

50 documents were classified as "Link" (Label 1).

27 documents were classified as "Co-occurrence" (Label 0).

Dataset Split: The 77 documents were split into a training set with 53 samples and a testing set with 24 samples.

Classical Model: This model used manually engineered features to make its predictions.

Quantum Model: This model used simple semantic features. It was executed on the ibm_brisbane quantum processor. The model was trained for 5 optimizer iterations, and the training log showed a fluctuating loss that started at 0.3290 and ended at 0.2824.

3. Results
The performance of both models on the 24-sample test set is detailed below.

```
Metric  Class   Classical Model (Engineered Features)   Quantum Model (Simple Semantic Features)
Accuracy        Overall 70.83%  62.50%
Precision       0 (Irrelevant)  1.00    0.00
1 (Relevant)    0.70    0.65
Recall  0 (Irrelevant)  0.12    0.00
1 (Relevant)    1.00    0.94
F1-Score        0 (Irrelevant)  0.22    0.00
1 (Relevant)    0.82    0.77
Support 0 (Irrelevant)  8 samples       8 samples
1 (Relevant)    16 samples      16 samples
```

4. Analysis and Interpretation

```
Classical Model Performance: The classical model achieved a respectable accuracy of 70.83%. However, its performance
was highly skewed. It successfully identified 100% of the relevant "Link" documents (recall=1.00). To do this, it
misclassified most of the irrelevant documents, only correctly identifying 12% of them (recall=0.12). This indicates
the model learned a strong bias to predict the majority class (Label 1).

Quantum Model Performance: The quantum model's accuracy was lower at 62.50%. Its classification report shows a complete
failure to identify any irrelevant documents, with precision and recall scores of 0.00 for Class 0. This performance is
statistically consistent with a strategy of simply guessing the majority class (Label 1) for every sample, which would
yield an accuracy of 16/24, or 66.7%. The model failed to learn a useful decision boundary from the simple semantic
features when faced with hardware noise and an imbalanced dataset.

5. Conclusion
The classical model with manually engineered features was the clear winner in this experiment. The key insight from this
result is that the quality and informational content of the features were the deciding factor. The sophisticated,
human-guided features used by the classical model provided a much stronger learning signal than the quantum feature map
was able to extract from simple semantic similarity scores. This result underscores the significant challenge of automated
feature extraction in the NISQ era and highlights the effectiveness of well-designed classical approaches."""
```

Out[23]:
```
'Experiment Report: A Comparison of Classical and Quantum Thematic Correlators\nDate: Thursday, August 21, 2025\nLocation: Bengaluru, Karnataka, India\nAuthor: Anirudh R\nProject Status: Complete\n\nExecutive Summary\nThis report details the results of a head-to-head experiment comparing a classical machine learning\nmodel with engineered features against a Variational Quantum Classifier (VQC) with simple semantic features.\nBoth models were tasked with identifying documents that describe a thematic link between two concepts\n("Immune System" and "Neurodegenerative Disease"). The dataset was dynamically generated from a MongoDB\ndatabase, resulting in an imbalanced set of 77 documents.\n\nThe results show that the classical model achieved a significantly higher accuracy (70.83%) compared to\nthe quantum model (62.50%). Analysis of the classification reports reveals that both models struggled with\nthe imbalanced dataset, but the classical model learned a useful, albeit biased, decision boundary. The\nquantum model, running on the ibm_brisbane hardware, failed to learn a meaningful pattern and resorted to\npredicting the majority class. The primary conclusion is that for this task, the manually engineered features\nprovided a much stronger signal than the quantum feature map was able to extract from simple semantic inputs.\n\n1. Experiment Objective\nThe goal was to test if a quantum re-ranker, using simple semantic features, could outperform a classical one\nthat required complex, manual feature engineering. The task was to classify documents as either describing a\nthematic link (Label 1) or merely containing a co-occurrence of keywords (Label 0).\n\n2. Methodology\nDataset Discovery: A dataset was dynamically created by querying a MongoDB database. This process discovered\n77 total candidate documents.\n\n50 documents were classified as "Link" (Label 1).\n\n27 documents were classified as "Co-occurrence" (Label 0).\n\nDataset Split: The 77 documents were split into a training set with 53 samples and a testing set with 24 samples.\n\nClassical Model: This model used manually engineered features to make its predictions.\n\nQuantum Model: This model used simple semantic features. It was executed on the ibm_brisbane quantum processor.\nThe model was trained for 5 optimizer iterations, and the training log showed a fluctuating loss that started\nat 0.3290 and ended at 0.2824.\n\n3. Results\nThe performance of both models on the 24-sample test set is detailed below.\n\nMetric\tClass\tClassical Model (Engineered Features)\tQuantum Model (Simple Semantic Features)\nAccuracy\tOverall\t70.83%\t62.50%\nPrecision\t0 (Irrelevant)\t1.00\t0.00\n1 (Relevant)\t0.70\t0.65\nRecall\t0 (Irrelevant)\t0.12\t0.00\n1 (Relevant)\t1.00\t0.94\nF1-Score\t0 (Irrelevant)\t0.22\t0.00\n1 (Relevant)\t0.82\t0.77\nSupport\t0 (Irrelevant)\t8 samples\t8 samples\n1 (Relevant)\t16 samples\t16 samples\n\n4. Analysis and Interpretation\nClassical Model Performance: The classical model achieved a respectable accuracy of 70.83%. However, its performance\nwas highly skewed. It successfully identified 100% of the relevant "Link" documents (recall=1.00). To do this, it\nmisclassified most of the irrelevant documents, only correctly identifying 12% of them (recall=0.12). This indicates\nthe model learned a strong bias to predict the majority class (Label 1).\n\nQuantum Model Performance: The quantum model\'s accuracy was lower at 62.50%. Its classification report shows a complete\nfailure to identify any irrelevant documents, with precision and recall scores of 0.00 for Class 0. This performance is\nstatistically consistent with a strategy of simply guessing the majority class (Label 1) for every sample, which would\nyield an accuracy of 16/24, or 66.7%. The model failed to learn a useful decision boundary from the simple semantic\nfeatures when faced with hardware noise and an imbalanced dataset.\n\n5. Conclusion\nThe classical model with manually engineered features was the clear winner in this experiment. The key insight from this\nresult is that the quality and informational content of the features were the deciding factor. The sophisticated,\nhuman-guided features used by the classical model provided a much stronger learning signal than the quantum feature map\nwas able to extract from simple semantic similarity scores. This result underscores the significant challenge of automated\nfeature extraction in the NISQ era and highlights the effectiveness of well-designed classical approaches.'
```

In [ ]: