SafeML Monitoring Systems on Quantum Machine Learning Models

Oliver Dunn

School of Computer Science
University of Hull

This dissertation is submitted for the degree of
Bachelor of Science

May 2025

I would like to dedicate this thesis to the late president carter, who inspired us all. . .

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 20,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Oliver Dunn
May 2025

Acknowledgements

And I would like to acknowledge The research done by Koorosh Aslansefat, Yiannis Papadopoulos, Ioannis Sokoros, Ramin Tavakoli Kolagari, and Declan Whiting in their proposal of the SafeML method for classifier safety monitoring, of which this project is an specialist augmentation.

Abstract

This paper showcases a potential implementation of a safety monitoring system, known as SafeML, onto hybrid quantum machine learning models built within a simulation of a quantum computing system. This implementation entails applying a selection quantum distance metrics onto the predictions of classifiers against true values, with the idea that this offers insight into model performance, upon which a layer of human monitored security can be added.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The safety monitoring of machine learning (ML) systems is critical, particularly in safety-sensitive domains such as healthcare, transportation, and energy production. Various methods exist to ensure ML safety, one of which is *SafeML*, which uses parametric distance measures to assess the proximity of a model's predictions to real test data, highlighting areas of classifier weakness that can be used to determine areas of uncertainty at or even before runtime.

SafeML was developed in response to the recent drastic increase in the adoption of machine learning classification solutions, as a proposed method of monitoring out-of-distribution detection (OODD) problems in such solutions.

A technology seeing similar growth in industry interest and general research is quantum computing, a new computing paradigm that leverages quantum mechanical phenomena such as uncertainty and superposition to perform calculations. Quantum computing holds promise in areas such as pharmaceutical development and financial modeling due to its ability to process vast amounts of data simultaneously, with immensely increased speed compared to classical systems [17].

As a result, a convergence of these technologies has been devised: *quantum machine learning* (QML), which seeks to harness the advantages of quantum computing for ML tasks such as classification and regression.

However, just as ML models must be adapted for quantum architectures, safety monitoring techniques designed for classical ML must also evolve to ensure compatibility with QML systems. Given the early stage of QML, research on safety monitoring in this context remains scarce. This gap raises concerns about the deployment of QML models without adequate safeguards, particularly given how susceptible quantum systems currently are to environmental conditions such as noise or background heat energy. Quantum computing requires very specific conditions to run properly as it operates on a very small scale—down to the electron—and so interference from even a minuscule amount of background thermal energy can introduce enough noise to obscure computations.

This project aims to propose a concept of quantum machine learning safety monitoring by expanding on the existing SafeML method in order to ensure its compatibility with quantum architecture and quantum-specific machine learning models. To do this, it will seek to apply quantum-specific statistical distance measures into the SafeML methodology flow, making adjustments to this methodology where necessary to ensure compatibility with quantum calculations.

1.1 Aims, Objectives, and Questions

1.1.1 Aims

This project's primary aim is to produce a modified methodology of the standard SafeML structure which gives special consideration to quantum data processing and the specific architecture of QML models. It will then apply quantum distance metrics to the results of these QML model predictions in a familiar way, showing the deviation of these results from model accuracy, ultimately providing a new method of safety monitoring and error detection in QML models.

1.1.2 Objectives

1. Source a range of quantum distance metrics.
2. Propose potential use cases for distance metrics in quantum machine learning model evaluation, and their direct applicability to the SafeMl method.
3. Use the formulas of chosen metrics to create reusable methods in the implementation language which will compute the distance of the quantum datasets.
4. Use distance methods to derive insights into model performance of a quantum machine learning solution.
5. Evaluate the efficiency of the Quantum SafeML approach in safety monitoring terms.

1.1.3 Questions

1. Does Quantum SafeML show potential for adequate safety monitoring of quantum machine learning systems?
2. Is Quantum SafeML functional on a range of different quantum machine learning implementations?
3. What areas of quantum machine learning benefit the most from a safety monitoring method such as Quantum SafeML?

1.2 Scope of the Study

This project aims to show a proof of concept of Quantum SafeML. However, given the lack of commercial availability of quantum hardware, it is not possible for this project to be undertaken using a real quantum computer. Due to this, a simulation of quantum hardware will be used. It is fully anticipated that such simulations do not provide completely accurate quantum functionality and will still be restricted by the capabilities of the classical machine running the simulation. This could mean that the quantum machine learning solutions built do not have the available quantum computing power to be effective models on the testing data and may perform worse than classical machine learning solutions. This project is not concerned with ensuring that quantum machine learning models built in quantum computing simulations are superior or even equal to standard classical models in the included implementations. Instead, the focus is on how the Quantum SafeML method is able to monitor the underperformance of these models.

1.2.1 Lack of Quantum Hardware

The primary constraint to our research will be the lack of availability of quantum hardware. Quantum computers require very specific conditions in order to run viably and are still in experimental testing phases in the laboratories of companies such as IBM and Google. It is not possible to test our implementation on a real quantum computer at this time, so quantum simulators will be explored.

There is a range of existing methods of simulating quantum hardware, many of which are open source. Notable examples include Qiskit, PennyLane, and Cirq. These will be explored further as decisions are made on the specific choices of this project's methodology and development.

1.2.2 Lack of Existing Safety Monitoring Resources for Quantum Systems

Perhaps directly tied to the constraint described above is the lack of current resources on quantum machine learning safety monitoring. This is of course due to the lack of industry application of quantum machine learning at this time, but just as its future use can be anticipated, the requirement of parallel safety monitoring methods can similarly be anticipated. However, no current proposal of ideal quantum safety monitoring yet exists, hence our methodology being largely centered around adapting classical methods as opposed to building a new method from the ground up.

1.2.3 Quantum Machine Learning Model Performance

The goal of this project is not to produce quantum machine learning models that outperform classical counterparts. In fact, given the constraints of quantum simulation, it is expected the models may underperform compared to conventional machine learning systems. The main concern is whether the statistical distance metrics employed are able to be applied as per the SafeML method to the results of these models.

1.2.4 Strictly Classification

While the SafeML method is speculated to work on a range of different types of machine learning problems, classification is chosen for this project to closely follow the initial SafeML proposed methodology.

1.3 Structure of the Thesis

This project begins with an introduction detailing the project, followed by a literature review covering classical machine learning, SafeML, quantum computing, and quantum machine learning. It then moves into the methodology, detailing how the implementation was conceived, researched, and designed. Following this, development will show how the implementation was built and tested. The evaluation section will discuss the results of the Quantum SafeML implementation. Finally, in the conclusion, the results will be used to draw a clear and concise basis for the quantum safety monitoring dialogue.

Introduction

In the introduction, a brief description of the context of the project is given, and the main goals of the project are outlined, giving explanation to the scope, limitations, and significance of the project.

Background

In the background, a general overview of the existing technologies this project will build upon will be given, detailing the idea of SafeML and the prospect of quantum machine learning.

Literature Review

The existing research into the technologies and principles necessary for this project will be explored in further detail, highlighting specific findings in order to form a solid foundation of the principles this method will build upon.

Research Gap and Problem Definition

Following the overview of the technologies given in the background, the existing gaps that prevent convergence of the technologies of SafeML and quantum machine learning are discussed. This extends to the general lack of quantum machine learning monitoring.

From this, the problem present in this domain will be definitively stated, and this will form the basis for how the Quantum SafeML method will be developed in order to address this issue.

Methodology

Here, the methodology through which the project will operate will be outlined, explaining specific choices made in implementation tools and expectations of the key steps involved. The experiments necessary to extract valuable information from existing implementations of the relevant disciplines will be defined, and the layout of how development and adaptation of these technologies will be approached with respect to the previously decided upon implementation tools.

Experiments

The Experiments chapter will showcase the technologies of the classical SafeML method, quantum simulation, and quantum machine learning in action, analyzing their functionality through the use of existing example work. This chapter will also explore the specific statistical distance measures Quantum SafeML will employ, ultimately using the findings of their formula analysis to define ideal use cases for each metric, selecting the metrics which align most closely with the Quantum SafeML method.

Development

Development will show the implementation of the chosen metrics into the selected technologies, and the specific classifier architectures which will form the example of the Quantum SafeML methodology in action.

Testing and Evaluation

Example QML models will be instantiated and fitted to given data. Then, the predictions of these models will be used to create two sets which will be passed to the implemented distance metrics. The values given by these metrics will be compared to overall model accuracy.

Results Discussion and Conclusion

The effectiveness of the proposed methodology will be discussed, and its implications for quantum machine learning safety monitoring concluded.

1.4 Project Significance

Quantum computing has been theorized to have clear advantages compared to classical systems in a wide range of applications. Some of these cases would be safety critical, such as the discovery of new drugs with quantum chemistry, and cybersecurity with quantum decryption.

With the theorized utility of quantum computing, it is clear that quantum machine learning will be a significant technology in the coming years, and an ever-growing dominance of machine learning solutions in a range of different fields shows a clear path forward where the two technologies will converge in order to harness the advantages of each for quantum machine learning solutions.

However, its efficiency will be proportional to the quality of the safety monitoring methods underpinning it, and this is an area that has yet to be fully explored.

This project will contribute to the dialogue of quantum machine learning safety monitoring, yielding results which can further the discussion of adequate quantum safety methodology as the technology continues to develop.

Chapter 2

Background

2.1 Classification in 1-Dimensional Data

Classification in probabilistic systems relies on analyzing the distributions associated with different classes. A simple example involves a one-dimensional dataset in which a classifier assigns an input feature x to one of two classes based on a threshold rule:

$$D(x) = \begin{cases} \text{Class } 1, & 0 < x \leq 100 \\ \text{Class } 2, & 100 < x \leq 200 \end{cases} \tag{2.1}$$

Each class is modeled by a probability density function (PDF), and classification decisions are made by comparing the likelihoods of x under each class distribution. Overlap between these distributions leads to misclassification.

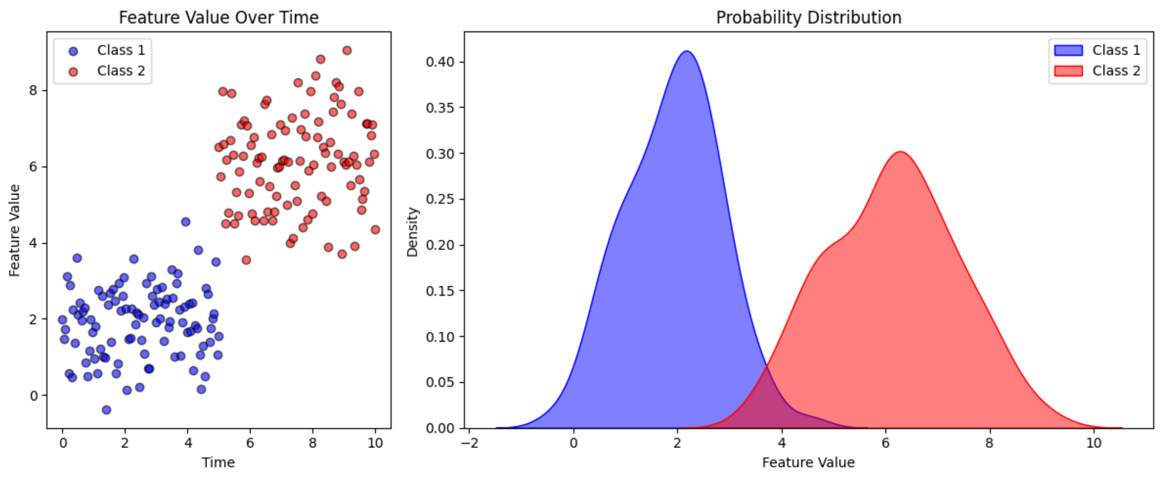


Figure 2.1 Basic probability distribution on an example 1-dimensional dataset

2.1.1 Classification Error and Conditional Risk

The total probability of classification error is given by integrating over the input space:

$$P(\text{error}) = \int_{-\infty}^{+\infty} P(\text{error}|x)P(x)\,dx \tag{2.2}$$

The conditional error probability is defined as the minimum of the posterior probabilities:

$$P(\text{error}|x) = \min[P(\text{Class } 1|x), P(\text{Class } 2|x)] \tag{2.3}$$

This framework reflects the fact that error arises in regions where class distributions overlap.

2.1.2 Bounding Error Using the Chernoff Bound

To bound the classification error, we consider two distributions $p_1(x)$ and $p_2(x)$, with general probabilities π_1 and π_2, respectively. The Chernoff bound provides an exponential upper bound:

$$P(\text{error}) \leq \min_{\lambda \in [0,1]} \left[\pi_1^{\lambda} \pi_2^{1-\lambda} \int_{\mathbb{R}} p_1(x)^{\lambda} p_2(x)^{1-\lambda} dx \right] \tag{2.4}$$

This bound becomes tighter as the overlap between distributions decreases. When $\lambda = 0.5$, the integral becomes the Bhattacharyya coefficient:

$$BC(p_1, p_2) = \int \sqrt{p_1(x)p_2(x)}dx \tag{2.5}$$

This yields the Bhattacharyya bound:

$$P(\text{error}) \leq \sqrt{\pi_1 \pi_2} \cdot BC(p_1, p_2) \tag{2.6}$$

2.1.3 Alternative Distance-Based Bounds

Other statistical distances can also be used to estimate classification error:

Total Variation Distance:

$$\delta_{TV}(p_1, p_2) = \frac{1}{2} \int |p_1(x) - p_2(x)|dx \tag{2.7}$$

This gives an upper bound:

$$P(\text{error}) \leq \min(\pi_1, \pi_2)(1 - \delta_{TV}(p_1, p_2)) \tag{2.8}$$

Kullback–Leibler Divergence: Although not symmetric, the KL divergence is often used in practice to compare distributions:

$$D_{KL}(p_1||p_2) = \int p_1(x) \log\left(\frac{p_1(x)}{p_2(x)}\right) dx \tag{2.9}$$

2.1.4 SafeML: A Framework for Monitoring Classification Safety

SafeML proposes comparing the predicted class distribution $\hat{p}(y|x)$ with a reference or true distribution using divergence measures:

$$\text{Divergence Score} = D(\hat{p}(y|x), p^*(y|x)) \tag{2.10}$$

When this divergence exceeds a threshold, a safety flag is raised. Distance measures include:

- Kullback–Leibler divergence
- Wasserstein distance
- Bhattacharyya distance

This approach enables the detection of anomalous or low-confidence predictions.

2.2 Quantum Classification

Quantum classification generalizes the above ideas to quantum systems, where states are described by density operators instead of real-valued vectors.

2.2.1 Quantum State Discrimination

Let ρ_1 and ρ_2 be density matrices representing two classes with priors π_1 and π_2. A binary measurement $\{M_1, M_2\}$ is used to decide between them.
The probability of error is:

$$P(\text{error}) = \pi_1 \text{Tr}(M_2 \rho_1) + \pi_2 \text{Tr}(M_1 \rho_2) \tag{2.11}$$

The optimal measurement minimizing (2.11) is given by the Helstrom measurement.

2.2.2 Quantum Chernoff Bound

The quantum analogue of the Chernoff bound is:

$$P(\text{error}) \leq \min_{\lambda \in [0,1]} \left[\pi_1^{\lambda} \pi_2^{1-\lambda} \mathrm{Tr}(\rho_1^{\lambda} \rho_2^{1-\lambda}) \right] \tag{2.12}$$

This trace expression replaces the classical integral. Fixing $\lambda = 0.5$ gives the quantum Bhattacharyya bound:

$$P(\text{error}) \leq \sqrt{\pi_1 \pi_2} \cdot \mathrm{Tr}(\sqrt{\rho_1}\sqrt{\rho_2}) \tag{2.13}$$

2.2.3 Quantum Distance Measures for Classification

In quantum settings, several operator distances help characterize distinguishability:

Trace Distance:

$$\delta_{\text{tr}}(\rho_1, \rho_2) = \frac{1}{2} \mathrm{Tr}|\rho_1 - \rho_2| \tag{2.14}$$

Bures Distance:

$$D_B(\rho_1, \rho_2) = \sqrt{2\left(1 - \mathrm{Tr}\left[\sqrt{\sqrt{\rho_1}\rho_2\sqrt{\rho_1}}\right]\right)} \tag{2.15}$$

Both distances satisfy metric properties and can be used as proxies for classification error, and these metrics specifically will be analyzed alognisde others in later stages fo experimentation for full implementation of this project's proposed method.

2.3 Latent Space Embeddings

2.3.1 Classical Embedding

Deep learning models often map inputs to a latent space $x \mapsto z \in \mathbb{R}^d$. In this space, distributions $p_1(z)$ and $p_2(z)$ can be analyzed using the same tools:

$$P(\text{error}) \leq \min_{\lambda \in [0,1]} \left[\pi_1^{\lambda} \pi_2^{1-\lambda} \int p_1(z)^{\lambda} p_2(z)^{1-\lambda} dz \right] \tag{2.16}$$

2.3.2 Quantum Latent Encoding

Quantum models may produce latent classical vectors or reduced density operators. If $\rho_j \mapsto p_j(z)$, and the encoder preserves distances, then:

$$P_{\text{latent error}} \leq \min_{\lambda \in [0,1]} \left[\pi_1^{\lambda} \pi_2^{1-\lambda} \int p_1(z)^{\lambda} p_2(z)^{1-\lambda} dz \right] \tag{2.17}$$

If the quantum geometry is preserved, this bound approximates (2.12).

2.4 Out-of-Distribution Detection (OODD)

OODD remains critical in both classical and quantum domains. It aims to detect when a sample lies outside the training distribution. Techniques include:

- Estimating low likelihood under the learned distribution.
- Comparing latent features to class centroids.
- Monitoring divergence measures via SafeML.

This mechanism is essential for safe deployment of machine learning systems.

2.5 Quantum Machine Learning Overview

Quantum Machine Learning (QML) utilizes quantum algorithms for classification, embedding, and inference. Models such as Variational Quantum Classifiers (VQCs) and Quantum Neural Networks operate on quantum circuits, producing probabilistic outputs post-measurement.

These models are sensitive to noise, requiring safety mechanisms in place to mittigate the effects of environmental disruptions.

2.6 Background Conclusion

Classification error can be rigorously analyzed through probabilistic distributions and their overlaps. Chernoff-type bounds apply both in classical and quantum domains. Quantum extensions use trace expressions in place of integrals and operator distance measures like the trace and Bures distances. The SafeML framework generalizes naturally to quantum models, offering a robust approach to error monitoring and OOD detection in future quantum systems.

Chapter 3

Literature Review

3.1 Introduction

This literature review explores the key technologies underpinning the implementation of Quantum Safe Machine Learning (SafeML), including quantum computing, quantum machine learning (QML), and classical SafeML. These foundational components collectively contribute to the development of secure, trustworthy artificial intelligence systems in a quantum context.

3.2 Quantum Computing: Fundamentals and Applications

Quantum computing is an alternative computing paradigm that has gradually progressed from theoretical concepts to early-stage implementations in recent years.

Unlike classical computers, which operate using binary bits (0 or 1), quantum computers utilize qubits—quantum units of information that leverage principles of quantum mechanics. These qubits can exist in multiple states simultaneously, offering a fundamentally different mode of computation [1]. Conceptually, a qubit represents the probabilistic spin state of a subatomic particle, which can be manipulated and measured to perform complex operations [2].

3.2.1 Fundamental Principles of Quantum Computing

Quantum systems require highly controlled environments to maintain stability. For example, IBM's quantum computers must operate at temperatures colder than outer space to reduce

thermal noise, which would otherwise destabilize the fragile quantum states [5].
The essential features that distinguish quantum computing from classical computing include:

- **Superposition**: Qubits can simultaneously represent both 0 and 1, enabling exponential scaling of computational states.
- **Entanglement**: Qubits can become entangled, allowing the state of one qubit to instantaneously affect another—regardless of distance—enabling faster and more efficient data correlations.
- **Quantum Interference**: Quantum systems leverage interference to reinforce correct computation paths while diminishing incorrect ones, which is critical in algorithmic decision-making.

3.2.2 Advancements in Quantum Computing

There is growing anticipation that quantum computers will surpass classical systems for specific types of problems. For instance, Google recently introduced a quantum chip named *Willow*, which reportedly completed a benchmark computation in five minutes—a task that would take classical supercomputers an estimated septillion years [3].

Despite such milestones, it is important to understand that quantum computers are not general-purpose replacements for classical machines. Their architecture is optimized for niche domains that benefit from parallel quantum state evaluations rather than sequential binary operations.

3.2.3 Applications of Quantum Computing

IBM has projected that chemistry will be among the first disciplines to benefit significantly from quantum computing. This is due to quantum systems' ability to simulate molecular structures and interactions with high fidelity—far beyond classical capabilities [2]. Such potential could accelerate research in materials science and pharmaceutical development.

Other impactful domains include:

- **Cryptography**: Quantum algorithms such as Shor's algorithm can factor large numbers efficiently, potentially breaking modern encryption schemes.
- **Optimization**: Quantum methods may offer faster solutions to complex optimization problems in logistics, transportation, and finance.

- **Artificial Intelligence**: Integrating quantum computing with AI could enable more powerful machine learning models and training efficiency.

3.2.4 Limitations and Challenges

Despite its potential, quantum computing faces several significant challenges:

- **Hardware Constraints**: Qubit stability demands extreme conditions, including cryogenic temperatures and isolation from environmental noise.
- **Error Correction**: Quantum operations are highly error-prone, requiring sophisticated error correction mechanisms to maintain computational fidelity.
- **Scalability**: Most current quantum processors are limited in qubit count and coherence time, restricting their use in large-scale real-world applications [5].

3.3 Quantum Computing Summary

This section has outlined the principles, advancements, applications, and limitations of quantum computing. While still emerging, quantum technologies hold significant promise for solving complex computational problems that are intractable for classical machines. Rather than replacing traditional systems, quantum computing is expected to complement them—particularly in specialized domains.
The following sections will delve into quantum machine learning and classical SafeML, highlighting their roles in enabling safer, more robust AI systems in a hybrid computing landscape.

3.4 Quantum Machine Learning (QML) Overview

Quantum Machine Learning (QML) is an interdisciplinary field combining quantum computing with machine learning methodologies. While a precise definition is still evolving, QML generally refers to one of two approaches: using classical machine learning to optimize quantum systems, or implementing machine learning algorithms on quantum hardware.
In this project, the focus is on the latter—constructing and deploying machine learning models that are intrinsically quantum in nature, leveraging quantum circuits and principles to perform classification and learning tasks.

3.4.1 Hybrid Quantum-Classical Models

A core focus of this project lies in hybrid quantum-classical machine learning models. These approaches combine the computational stability and scalability of classical architectures with the probabilistic and high-dimensional capabilities of quantum systems. By integrating both paradigms, hybrid models aim to address key challenges in quantum computing, such as quantum noise and hardware limitations, while harnessing the benefits of quantum parallelism [6].

Due to the current limitations of quantum hardware, many hybrid models are implemented using quantum simulators. While these simulators cannot fully replicate the capabilities of real quantum systems, they offer an effective approximation and serve as a crucial bridge for evaluating quantum algorithms before physical deployment.

3.4.2 Quantum Neural Networks (QNNs)

Quantum Neural Networks (QNNs) represent a quantum extension of traditional neural networks. These architectures utilize quantum circuits and measurement operations instead of classical activation functions to process and learn from data. At their core, QNNs consist of parameterized quantum circuits—sometimes called variational circuits—that are trained to approximate complex mappings from input to output.

Emerging research suggests that QNNs may provide advantages in tasks such as quantum state classification and generative modeling [7]. However, due to quantum hardware constraints, including limited qubit counts and decoherence, the scalability and efficient training of QNNs remain active areas of investigation.

3.4.3 Variational Quantum Classifiers (VQCs)

Variational Quantum Classifiers (VQCs) are another promising approach in the QML landscape. These models employ parameterized quantum circuits optimized through classical techniques (e.g., gradient descent) to perform classification tasks. A typical VQC pipeline includes three main stages:

- **Feature Encoding**: Input data is transformed into quantum states via encoding circuits.
- **Entangling Layers**: Qubits are manipulated through entanglement to model complex data relationships.

- **Measurement**: The resulting quantum state is measured and interpreted classically to determine class probabilities.

Although VQCs have shown promising results in small-scale classification tasks [8], they face similar limitations to other QML models—namely noise, hardware instability, and challenges with convergence in training.

3.5 Quantum machine learning Summary

This section has explored foundational concepts in Quantum Machine Learning, focusing on hybrid quantum-classical models, Quantum Neural Networks (QNNs), and Variational Quantum Classifiers (VQCs). While the field is still in its early stages, ongoing research continues to push the boundaries of what is computationally possible using quantum-enhanced models. The next section introduces classical SafeML approaches that aim to ensure machine learning models behave safely and reliably in uncertain or high-risk scenarios.

3.6 Safety Monitoring in Machine Learning

Ensuring the safe deployment of machine learning systems is critical in applications where incorrect predictions can have serious consequences. In safety-critical domains such as healthcare, finance, or autonomous systems, it is essential that models not only perform well on familiar data but also respond appropriately to unexpected or novel situations. This section reviews two prominent safety monitoring methods: **Out-of-Distribution (OOD) Detection** and **SMARLA**—a framework designed specifically for Deep Reinforcement Learning (DRL) agents.

3.6.1 Out-of-Distribution (OOD) Detection

Out-of-Distribution detection is a vital mechanism for identifying inputs that deviate from the training distribution. When machine learning models are exposed to such unfamiliar inputs, their predictions become unreliable. OOD detection techniques aim to flag these instances, allowing for human intervention or alternative decision pathways.
One widely studied method is **ODIN** (Out-of-Distribution detector for Neural networks), which enhances softmax-based classification models by adjusting temperature scaling and applying small input perturbations. This improves the model's ability to distinguish between in-distribution and OOD data [10].

OOD detection plays a particularly important role in real-time systems. For instance, in autonomous driving, encountering a road condition or object not seen during training could be catastrophic if misclassified. OOD mechanisms serve as a first line of defense against such unpredictable scenarios.

3.6.2 SMARLA: A Safety Monitoring Approach for Deep Reinforcement Learning Agents

SMARLA (Safety Monitoring for Deep Reinforcement Learning Agents) is a black-box safety monitoring technique designed for environments where agents learn through reinforcement. While DRL agents excel at maximizing cumulative reward, they often do so without incorporating safety constraints, which can result in unsafe behavior.

SMARLA addresses this challenge by predicting potential safety violations during the agent's execution. It relies on observing the agent's Q-values—estimations of expected future rewards for state-action pairs—and uses **state abstraction** to simplify the agent's operational space. This abstraction enhances the model's predictive power by generalizing the detection of dangerous states.

Through empirical validation on benchmark DRL environments, SMARLA has demonstrated an ability to predict violations early—typically halfway through execution—with a low false positive rate. The framework also supports adjustable decision criteria, using confidence intervals on violation probabilities to balance early detection with false alarm minimization.

3.7 Safety monitoring Summary

In this section, two leading safety monitoring approaches were examined. OOD detection focuses on identifying unfamiliar data that may lead to incorrect classifications, while SMARLA offers a dynamic monitoring system for DRL agents by predicting safety violations before they occur. Together, these approaches contribute to a broader framework for ensuring the safety, reliability, and trustworthiness of machine learning systems in real-world and safety-critical applications.

3.8 SafeML Overview

SafeML is a proposed framework for safety monitoring in machine learning (ML) applications, particularly within safety-critical domains. It utilizes statistical distance measures to quantify discrepancies in model predictions, allowing for the identification of potential issues. Based on these discrepancies, SafeML determines whether human intervention is necessary, playing a critical role in mitigating risks associated with model uncertainty and ensuring robust decision-making in AI systems [12].

The demand for safety monitoring has surged as AI systems are increasingly deployed in high-stakes environments such as healthcare, finance, and autonomous systems. Recent studies highlight the necessity for advanced reliability engineering techniques to enhance the safety of AI-driven systems [13].

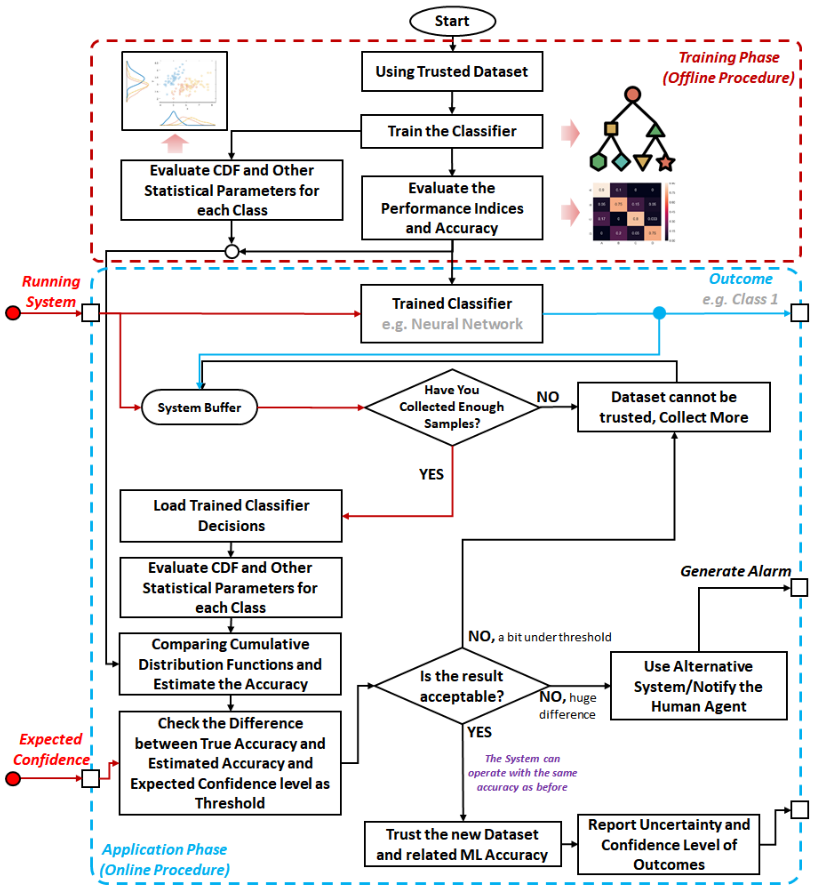

Start
Training Phase (Offline Procedure)
Using Trusted Dataset
Train the Classifier
Evaluate CDF and Other Statistical Parameters for each Class
Evaluate the Performance Indices and Accuracy
Running System
Outcome e.g. Class 1
Trained Classifier e.g. Neural Network
System Buffer
Have You Collected Enough Samples?
NO
Dataset cannot be trusted, Collect More
YES
Load Trained Classifier Decisions
Evaluate CDF and Other Statistical Parameters for each Class
Comparing Cumulative Distribution Functions and Estimate the Accuracy
Check the Difference between True Accuracy and Estimated Accuracy and Expected Confidence level as Threshold
Expected Confidence
Generate Alarm
NO, a bit under threshold
Is the result acceptable?
NO, huge difference
Use Alternative System/Notify the Human Agent
YES
The System can operate with the same accuracy as before
Trust the new Dataset and related ML Accuracy
Report Uncertainty and Confidence Level of Outcomes
Application Phase (Online Procedure)

Figure 3.1 SafeML proposed method. Source: [12].

3.8.1 The Risks of Poor Machine Learning Applications

Inadequately applied machine learning can introduce significant risks, particularly when models are deployed without sufficient validation. Research has highlighted challenges at various stages of the ML deployment workflow, including data management, model selection, training, verification, and maintenance [14]. Issues such as poorly curated datasets, improper hyperparameter tuning, and mismatched model architectures can compromise performance and reliability. These challenges underscore the growing need for safety monitoring approaches like SafeML.

3.8.2 SafeML as an Out-of-Distribution (OOD) Detection Mechanism

SafeML can function as a form of Out-of-Distribution (OOD) detection by identifying when incoming data deviates from the training data distribution. OOD detection is crucial for AI safety, as it helps prevent unpredictable or unreliable model behavior when confronted with unfamiliar data. Recent research has explored OOD detection across various ML models and methodologies, reinforcing its role in enhancing system robustness [15]. SafeML employs statistical distance metrics to assess whether incoming data distributions significantly differ from those observed during training.

3.8.3 The Role of Statistical Distance Metrics

SafeML uses several statistical distance measures, such as the Anderson-Darling test, Wasserstein distance, and Kuiper's test, to evaluate discrepancies between predicted and actual results. These metrics help determine whether the system's confidence in its predictions remains within an acceptable range. If significant deviation is detected, an alert is triggered for human oversight [12].
Below, a brief analysis is provided for each metric, including the applied formulas, a description of how they work, and their potential role in identifying differences between classifier predictions and true values.

Anderson-Darling Test

The Anderson-Darling test statistic A^2 quantifies the discrepancy between an empirical cumulative distribution function (CDF) and a theoretical CDF, with greater sensitivity to the tails of the distribution. This makes it particularly useful for identifying outliers or extreme predictions that could signal unsafe behavior by the model.

$$A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} ((2i-1)(\ln F(X_i) + \ln(1 - F(X_{n+1-i})))) \tag{3.1}$$

where:

- n: Sample size.
- $F(X_i)$: Cumulative probability of the i-th ordered sample value under the theoretical distribution.
- X_{n+1-i}: The $(n+1-i)$-th ordered sample value from the end.

This metric can detect discrepancies between predicted and true distributions, particularly when predictions deviate significantly from expected values. In SafeML, it helps identify unsafe predictions that fall outside acceptable thresholds, such as extreme or unlikely outcomes.

Wasserstein (Earth Mover's) Distance

The Wasserstein distance, or Earth Mover's distance, measures the "work" required to transform one distribution into another. It is useful for quantifying the difference between predicted and true distributions, especially when comparing distributions with different supports.

$$W(F,G) = \int_{-\infty}^{\infty} |F(x) - G(x)| \, dx \tag{3.2}$$

where:

- $F(x)$ and $G(x)$: Cumulative distribution functions (CDFs) of the two distributions.
- $|F(x) - G(x)|$: Absolute difference between the CDFs at each point x.

In SafeML, the Wasserstein distance quantifies how much the predicted distribution diverges from the true one. A larger distance suggests significant discrepancies that could signal unsafe behavior or poor model generalization.

Kuiper Distance

The Kuiper distance captures both positive and negative deviations symmetrically, making it well-suited for cyclic data. It contrasts with metrics like the Kolmogorov-Smirnov distance by treating overestimates and underestimates equally.

$$V = D^{+} + D^{-} \tag{3.3}$$

where:

- $D^{+} = \sup_x (F(x) - G(x))$: Maximum positive deviation between the CDFs.
- $D^{-} = \sup_x (G(x) - F(x))$: Maximum negative deviation between the CDFs.

In SafeML, the Kuiper distance is valuable for detecting situations where the model's predictions are systematically overestimated or underestimated, particularly in safety-critical contexts where balanced predictions are crucial.

Kolmogorov-Smirnov (KS) Distance

The Kolmogorov-Smirnov distance D measures the maximum absolute difference between two CDFs, identifying the point of greatest divergence between predicted and true distributions.

$$D = \sup_x |F(x) - G(x)| \tag{3.4}$$

where:

- $F(x)$ and $G(x)$: Cumulative distribution functions of the two distributions.
- $\sup_x |F(x) - G(x)|$: Maximum vertical difference between the CDFs.

For SafeML, the KS distance effectively flags significant deviations between predicted and true distributions, which may indicate unsafe predictions in safety-critical applications.

Cramér-von Mises Distance

The Cramér-von Mises distance measures the overall discrepancy between two distributions by summing the squared differences across all points, providing an aggregate measure of prediction accuracy.

$$W^2 = \int_{-\infty}^{\infty} (F_n(x) - G_m(x))^2 \, dF(x) \tag{3.5}$$

where:

- W^2: The Cramér-von Mises statistic, representing the overall squared difference between the CDFs.
- $F_n(x)$ and $G_m(x)$: Empirical CDFs of the two samples.
- $(F_n(x) - G_m(x))^2$: Squared difference between the CDFs at each point x.

In SafeML, the Cramér-von Mises distance helps identify overall discrepancies across the prediction range, offering insights into systematic deviations that could indicate unsafe behavior.

3.8.4 The Importance of Human Oversight

It is widely acknowledged that AI systems should incorporate human oversight, particularly in safety-critical applications. Over-reliance on automation without human intervention can lead to unintended errors and ethical concerns. Regulatory frameworks, such as the *EU AI Act*, emphasize the need for human-in-the-loop mechanisms to ensure AI systems remain accountable and aligned with ethical standards [16]. SafeML supports this principle by ensuring that model behavior anomalies are escalated to human operators for validation.

3.8.5 Implementation of SafeML in ML Pipelines

SafeML can be integrated into machine learning workflows through a structured process. The system first gathers predictions from a classifier and computes statistical distances between predicted and actual results. A system buffer ensures sufficient data is collected before classification. If the statistical discrepancy exceeds a predefined threshold, SafeML triggers an alert for human intervention. The adaptability of this approach allows SafeML to be implemented in various ML architectures while maintaining its core functionality [12].

3.9 SafeML Summary

The integration of SafeML principles into quantum machine learning (QML) systems presents critical safety challenges. While quantum computing offers significant potential for enhancing machine learning tasks, ensuring the reliability, safety, and interpretability of quantum-enhanced models remains an open problem. The unique characteristics of quantum computing, such as hardware noise, decoherence, and the lack of standardized benchmarking techniques, introduce new obstacles that classical SafeML frameworks were not initially designed to address.

One key challenge in this domain is the verification and validation of quantum-enhanced ML models. While classical systems already struggle with issues such as adversarial robustness, fairness, and explainability, these concerns are exacerbated in QML due to quantum-specific factors. To address these concerns, hybrid monitoring approaches that combine classical techniques with quantum-specific characteristics will be essential in developing Quantum SafeML. Potential solutions include Quantum Neural Networks (QNNs) and Variational Quantum Classifiers (VQCs) [7], which can be used to explore how quantum circuits behave under different conditions.

As quantum computing hardware evolves, the development of rigorous safety monitoring frameworks and quantum-specific testing methods will be critical for ensuring QML systems are both reliable and safe. Future research should focus on bridging the gap between quantum computing and machine learning, with an emphasis on safety and robustness to support responsible and secure adoption of quantum technologies in AI-driven applications.

Chapter 4

Research Gap and Problem Definition

4.1 Research Gap

While previous research in the SafeML methodology provides a solid way of monitoring classifier safety in classical systems, and early designs of quantum machine learning architecture have shown promise, the marriage of these technologies remains to be seen. The key gaps include:

- The lack of a well-defined modification to the SafeML method for quantum machine learning solutions.
- Existing error estimation methods for quantum machine learning models are limited.
- There is no established method for evaluating false positives and false negatives in quantum classifiers using empirical cumulative distribution functions.
- While quantum distance metrics exist, these have yet to be employed in a safety monitoring capacity.

This research aims to bridge these gaps by extending the error estimation protocols present in the SafeML method to quantum computing, leveraging quantum statistical distances, and demonstrating how classical safety monitoring methods can be adapted to suit quantum environments. Additionally, it will explore the impact such monitoring layers can have on overall quantum classifier safety.

4.1.1 Why Classical SafeML is Inadequate for Quantum Machine Learning

The SafeML method attempts to mitigate the effect of classifier confusion by providing an added layer of model monitoring. It does this by employing a selection of statistical distance measures on sets of predictions from a trained classifier, and the results of these are compared to the true accuracy of the classifier to determine if certain predictions are made in confusion and might be unsafe.

Unlike classical systems, quantum computing systems differ in fundamental ways:

1. **Quantum data representation:** Quantum data does not have a deterministic representation but exists in a superposition of states, requiring representation in formats such as density matrices, which index probabilities of outcomes.

2. **Quantum machine learning model architecture:** Quantum machine learning models are generally built with this representation in mind and take in and output in types and formats that are probabilistic and uncertain in nature.

3. **Distance metric incompatibility:** SafeML uses a range of classical distance metrics that will not directly compute on quantum-encoded data. Metrics such as Kolmogorov-Smirnov and Wasserstein distance will not be effective at calculating the confidence in quantum machine learning classifier predictions. Therefore, consideration must be given to similar formulas that can operate on quantum data.

Given these challenges, adapting the SafeML method to quantum systems requires exploring quantum analogs to the traditional distance measures it typically uses. Analysis and evaluation of these formulas must be conducted to determine if they would fit the use case of quantum machine learning classification and, if so, whether they can be implemented in a deployable Quantum SafeML solution.

4.2 Problem Statement and Mathematical Formulation

This project aims to develop a quantum-adapted version of the SafeML method by refactoring the flow of SafeML to operate with quantum-compatible distance metrics, which are then applied to quantum machine learning model architectures. Specifically, it seeks to answer:

> *"Can the SafeML method effectively lend itself to quantum computing technology, provided adequate adaptations are made to encompass the differing characteristics of quantum computing and quantum machine learning?"*

This problem specifically entails ensuring adequate data formatting into density matrices, a fundamental concept in quantum mechanics that represents the state of a quantum system. A density matrix ρ can be written as a weighted sum of outer products of quantum states:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \tag{4.1}$$

where:

- p_i is the probability associated with the quantum state $|\psi_i\rangle$,
- $|\psi_i\rangle$ is the i-th pure state of the quantum system.

This format illustrates how quantum data encoding differs from classical data, and it will be fundamental in bridging the gap between classical SafeML and a quantum-based solution.

4.3 Scope and Limitations

This research focuses on classification models, with various model architectures to be explored. It has been noted that the general structure of the SafeML method could also lend itself to regression models and more complex types of machine learning. However, for the purposes of this research, only classification problems will be considered when developing and evaluating the Quantum SafeML method.

4.4 Research Gap Summary

This chapter has outlined the research gap in quantum machine learning safety monitoring systems, highlighting the limitations of the current SafeML method in this domain. It has also presented the problem statement for adapting SafeML to quantum machine learning. The next chapter will provide a comprehensive review of existing technologies and example implementations, leading to a proposed design for the quantum SafeML method.

Chapter 5

Methodology

5.1 Methodology Overview

The focus of this project is to provide a proof of concept for adapting the SafeML method to quantum machine learning implementations using quantum distance metrics.

The objectives involved in this process include sourcing a range of quantum distance metrics and analyzing them for their key potential use cases in fully realized quantum machine learning systems. Once these metrics have been identified, the underlying mathematical formulas used to compute each metric will be converted into a form of pseudocode, approximating the structure of their implementation in a programming context. These will then be implemented in a chosen programming language to interact with a model built within a quantum machine learning system. The metrics will be applied to the implemented model, ultimately yielding results that highlight the key differences between predictions and actual values.

The strategy for developing Quantum SafeML will largely be based on experimenting with different datasets passed to selected quantum machine learning implementations, and calculating the values of the selected statistical distance metrics between the models' predictions and the real values withheld during the testing phase of classifier evaluation.

5.2 Implementation Choices

The specific implementation choices for this project include: the programming language used, the quantum computing framework selected to simulate a quantum system, the particular quantum machine learning models utilized for testing and evaluating the Quantum SafeML method, and the specific quantum distance measures chosen for implementation.

5.2.1 Language

Many different languages are used for machine learning classification. However, for the purposes of this project, special consideration must be given to languages that already feature implementations of the SafeML method, as this will make it easier to translate the method into our own environment.

Furthermore, consideration must be given to which languages have the necessary infrastructure for simulating quantum computing systems. This will be done via specific libraries and frameworks, so a language compatible with a wide range of options in this domain is preferable.

Existing SafeML work includes example implementations in Python, R, and MATLAB. Each of these languages has its own benefits and drawbacks in isolation; however, among the three, only Python is widely compatible with quantum computing libraries. As a result, it is the clear choice for this project's implementation.

5.2.2 Quantum Framework

As discussed previously, quantum computing hardware is not available for this project. Therefore, simulating quantum computing technology is essential. To achieve this, a range of available frameworks will be analyzed to identify the best option for our needs.

Quantum computing frameworks differ in functionalities, optimizations, and hardware compatibility. This section compares four major frameworks—Qiskit, PennyLane, Cirq, and QuTiP—discussing the benefits and drawbacks of each, before concluding with a chosen framework for this project.

Qiskit (IBM)

Qiskit is an open-source quantum computing framework developed by IBM. It is used to construct quantum circuits, algorithms, and environments that can be run on IBM's real quantum hardware, albeit currently with limited capacity.

Benefits	Drawbacks
Well-Supported: Backed by IBM with extensive documentation and tutorials.	**IBM-Centric**: Primarily designed for IBM's hardware, limiting interoperability with other systems.
Hardware Access: Provides direct access to IBM's real quantum computers and simulators.	**Steep Learning Curve**: More low-level than some alternatives; requires understanding of quantum gates and circuits.
Comprehensive: Offers tools for circuit design, noise modeling, and quantum algorithms, with a dedicated quantum machine learning library.	**Recent Migration**: A recent update has made some resources outdated, though the community remains active in addressing issues.
Python-Friendly: Designed for Python developers with an intuitive API.	

Table 5.1 Benefits and Drawbacks of Qiskit

PennyLane

PennyLane is a quantum machine learning framework optimized for hybrid quantum-classical computing and variational quantum algorithms.

Benefits	Drawbacks
Hybrid Quantum-Classical Computing: Designed for variational quantum algorithms and QML.	**Circuit Design is Less Intuitive**: Compared to Qiskit and Cirq, it uses a more abstract, function-based design.
Machine Learning Integration: Works seamlessly with TensorFlow and PyTorch.	**Not Optimized for Low-Level Quantum Experiments**: Focused on QML, making it less suitable for general quantum algorithm development.
Strong Auto-Differentiation: Enables efficient gradient-based optimization of quantum circuits.	

Table 5.2 Benefits and Drawbacks of PennyLane

Cirq (Google)

Cirq is an open-source framework developed by Google, designed for detailed control over quantum circuits.

Benefits	Drawbacks
Optimized for Google Quantum Processors: Specifically built for Google's quantum hardware.	**Limited Hardware Support**: Focused on Google processors, reducing compatibility.
Low-Level Control: Offers fine-tuned access to individual qubits and operations.	**Less Beginner-Friendly**: Requires deep quantum computing knowledge.
Efficient Simulation: Designed for fast classical simulation of quantum circuits.	**Smaller Community**: Fewer resources and user contributions than Qiskit.
Open-Source and Extensible: Easily integrated with other frameworks.	

Table 5.3 Benefits and Drawbacks of Cirq

5.2.3 QuTiP (Quantum Toolbox in Python)

QuTiP is a Python framework primarily focused on simulating quantum systems, especially open quantum systems and quantum dynamics.

Benefits	Drawbacks
Great for Quantum Simulations: Particularly suited for modeling quantum dynamics and open systems.	**No Direct Hardware Execution**: Not designed to interface with real quantum computers.
Mathematical and Physics-Oriented: Strong tool for theoretical and academic exploration.	**Not Focused on Quantum Algorithms**: Less ideal for practical quantum computing or machine learning.
Flexible and Customizable: Ideal for in-depth study of non-circuit-based quantum mechanics.	

Table 5.4 Benefits and Drawbacks of QuTiP

5.2.4 Why Qiskit is Chosen

Qiskit is selected as the preferred quantum computing framework due to its active support community—an essential resource given the experimental nature of this project.

While PennyLane is specialized for quantum machine learning, Qiskit features a dedicated quantum machine learning library with functionality suited to the goals of this project, including support for hybrid models.

Despite a steeper learning curve resulting from a recent migration, these updates have made Qiskit more robust and future-proof, ensuring that this implementation of the Quantum SafeML method is aligned with the current state of quantum simulation technology.

- Provides access to real IBM quantum hardware for potential future experimentation.
- Offers strong community support and extensive documentation.
- Includes simulation and noise modeling tools for realistic testing.
- Supports a wide range of quantum algorithms, enhancing flexibility.

5.2.5 QML Models Chosen

Due to the limitations of quantum simulation, model selection must reflect what can be realistically executed within a classical computing environment. One strategy for mitigating these constraints is to use hybrid quantum-classical machine learning models.

These models combine the strengths of classical and quantum architectures. The classical components help reduce computational overhead and facilitate data translation from classical formats into quantum states.

Since SafeML is currently a fully classical method with no quantum-native equivalent, applying it to hybrid models presents an ideal bridge between classical and quantum domains.

5.2.6 Variational Quantum Classifier

Variational Quantum Classifiers (VQCs) are a type of hybrid model. They encode classical data into quantum states using a feature map, and then process it through a variational quantum circuit—a quantum circuit with tunable parameters.

Like classical models, these parameters are optimized using a loss function to improve classification accuracy. The circuit produces a probability distribution over potential classes. Classical post-processing is used to interpret this distribution and produce the final class prediction (usually the class with the highest probability).

VQCs benefit from quantum principles such as entanglement, allowing them to represent complex relationships between data features. Their hybrid structure makes them particularly suited for simulated quantum environments running on classical hardware.

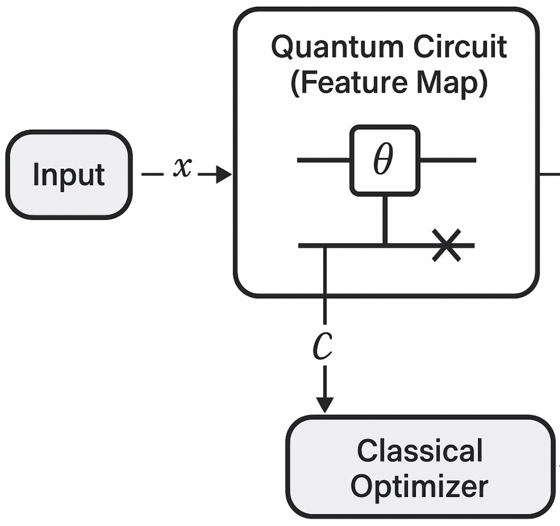


Figure 5.1 VQC diagram showing how inputs are passed to a feature map, drawn by myself

5.2.7 Quantum Neural Network

Quantum Neural Networks (QNNs) are quantum analogues of classical neural networks. Classical neural networks attempt to replicate the structure of the human brain by assigning weights to connections between layers of nodes, thereby learning patterns in the data.

QNNs elevate this concept by using parameterized quantum circuits in place of classical layers. Classical data is encoded into a quantum state using a feature map. These quantum states are then processed to evaluate the probabilities of different class labels.

The Quantum Convolutional Neural Network

The Quantum Convolutional Neural Network (QCNN) is the specific QNN architecture selected for this project. Like its classical counterpart, it alternates between convolutional and pooling layers—but on a quantum circuit.

QCNNs differ from classical CNNs by using quantum circuits and feature maps to encode input data. They use qubits to probabilistically learn data patterns, enabling a potentially more nuanced understanding of the dataset's features.

5.3 Experiments

This phase of the project involves foundational research into the core concepts underpinning Quantum Safe ML, setting the stage for its eventual implementation. Specifically, this includes an investigation into quantum distance metrics, a detailed analysis of the classical SafeML method, the instantiation of a quantum environment, and the implementation of quantum machine learning (QML) models.

5.3.1 Analysis of Quantum Distance Metrics

A key component of this project is the evaluation of various quantum distance metrics, which are essential for developing Quantum Safe ML. Unlike classical metrics that operate on deterministic data, quantum metrics are tailored for probabilistic data represented by density matrices. Consequently, specialized measures such as trace distance and matrix multiplication are required. Classical distance metrics commonly used in Safe ML are not directly transferable to quantum systems, making the identification and analysis of quantum-specific alternatives a critical task.

The exact selection of metrics will depend on what is available in the current literature, as quantum distance metrics remain an evolving field. Nevertheless, the project will aim to include a representative set for comparison—including those like the Wasserstein distance, which has been adapted for quantum contexts.

Each chosen metric will be examined in detail, with attention given to recurring mathematical structures (e.g., the use of the trace operator), enabling the identification of reusable computation steps. This analysis will then be translated into flowcharts or pseudocode to outline the metric calculation procedures, which will aid in designing a modular and efficient implementation.

Additionally, the investigation will consider the applicability of each metric to specific QML scenarios, particularly within quantum-classical hybrid models. Finally, the implementation feasibility of each metric will be assessed based on factors such as computational complexity, suitability for quantum architectures, and relevance to the goals of Quantum Safe ML.

5.3.2 Analysis of the Safe ML Method

To adapt SafeML to the quantum domain, a comprehensive understanding of the original classical method is necessary. Fortunately, the creators of SafeML have provided detailed example notebooks demonstrating its application to classification problems.

Safe ML Method Adaptation

The classical Safe ML approach will be evaluated in environments such as Python, R, and MATLAB to determine how its components—particularly its use of statistical distance metrics—can inform the development of Quantum Safe ML. Although Safe ML can be applied to regression, it is primarily oriented toward classification tasks, and this project will follow that focus.

The goal is to construct hybrid models that operate on classification data while being compatible with quantum computation. The feasibility and performance of such models in quantum environments will be critically assessed.

Safe ML Method Resources

The foundational reference for this analysis is the paper *"SafeML: Safety Monitoring of Machine Learning Classifiers through Statistical Difference Measure"*, which is supported by an official code repository. This resource includes examples of Safe ML in action and provides playground datasets for reproducibility. While alternative implementations exist, the general method described in the original paper will serve as the primary template for quantum adaptation.

Methodological Analysis

The Safe ML method will be dissected through a step-by-step walkthrough of the provided notebooks. These examples, which include the application of a neural network to the German traffic sign dataset, offer a practical foundation for identifying the necessary transformations to accommodate quantum elements.

Implementation Decisions and Considerations

Following the walkthrough, the implementation choices of the original Safe ML method will be analyzed. These insights will inform which aspects can be preserved in a quantum context and which require adaptation due to fundamental differences in quantum computing.

Anticipated Modifications

While many required changes will become clear during the analysis, several adaptations are already anticipated:

- Integration with quantum classifier architectures.
- Compatibility with quantum-encoded data formats.
- Accommodation of the mathematical and computational complexity of quantum distance metrics.

These modifications are essential to make Safe ML feasible in quantum environments and are aligned with the overarching aim of developing a robust Quantum Safe ML methodology.

5.3.3 Quantum Environment Instantiation

To test and develop QML models, a simulated quantum environment will be created using Qiskit. This emulation provides a platform where models can be developed, tested, and evaluated with high fidelity to real quantum hardware.

This environment will be constructed by following official Qiskit documentation and example notebooks, which provide step-by-step guidance for quantum circuit design, simulator usage, and backend configuration. A comprehensive simulation will ensure the accurate representation of quantum phenomena such as entanglement and superposition, both of which are critical for QML applications.

Special care will be taken to include all foundational elements of a quantum system, including quantum gates, circuit composition, and backend parameters, to support scalable experimentation.

5.3.4 Quantum Machine Learning (QML) Model Instantiation

The project will implement and study two prominent QML models: Variational Quantum Classifiers (VQCs) and Quantum Convolutional Neural Networks (QCNNs). Detailed examples for both models are available in Qiskit's documentation and will be used as a foundation.

By following these examples closely, the project will establish the necessary workflows for:

- Initializing quantum circuits specific to QML models.
- Encoding data into quantum states.
- Executing quantum training loops using parameterized circuits.

This process will build the required expertise and framework for applying distance metrics to model predictions.

Note on Model Accuracy

It is important to note that the objective of this project is not to surpass classical machine learning models in terms of raw accuracy. While quantum advantage is demonstrable in some specific tasks, QML models currently do not exhibit significant superiority over classical models in general-purpose classification tasks on standard datasets.

Thus, accuracy will serve only as a reference point for comparing metric performance rather than being a success criterion itself. The focus will be on ensuring that:

- Functional QML models can be successfully instantiated.
- These models can demonstrate some level of classification capability.
- The environment supports experimentation with quantum distance metrics.

By meeting these criteria, the groundwork will be laid for assessing the viability of Quantum Safe ML in real-world settings as quantum hardware continues to advance.

5.4 Development

5.4.1 Developing Distance Functions

Using the designed algorithms for the chosen distance metrics, these will be implemented in the selected programming language. For this project, the implementation language is Python. However, the initial pseudocode designs will be closely followed to maximize the potential for this work to be easily adapted to other environments, should appropriate quantum frameworks emerge for languages such as R or MATLAB in the near future.

5.4.2 Unit Tests for Distance Functions

A series of unit tests will be constructed specifically for the composition of the distance functions. These tests will then be deployed and evaluated. The unit tests are intended to assess the basic logical functionality of each function, ensuring that they are logically sound and robust to errors before being applied to quantum machine learning problems.

Unit Test Cases for Distance Functions

The following unit test cases will be used to validate the correctness and reliability of the implemented distance functions:

- `test_identical_matrices`
  Ensures that the respective distance metric returns **0** for identical matrices.
- `test_mismatched_shapes`
  Ensures that an **error is raised** when input matrices have **different shapes**.
- `test_range_validity`
  Ensures that the computed distance is within the valid range, typically $[0,1]$.
- `test_large_matrices`
  Tests the behavior of the distance metric with **larger matrices**, ensuring scalability.
- `test_symmetric`
  Confirms the **symmetry** of the metric: $D(A,B) = D(B,A)$.
- `test_mismatched_trace`
  Ensures that an **error is raised** if input matrices have a trace **not equal to 1**.
- `test_positive_semidefinite`
  Ensures that an **error is raised** if matrices are **not positive semidefinite**.
- `test_quantum_specific_objects`
  Confirms compatibility with **quantum-specific objects** such as Qiskit's `DensityMatrix`.
- `test_different_matrices`
  Ensures that the metric returns a **positive value** for non-identical matrices.

5.4.3 Implementing Functions into the SafeML Method

Once the functions have been verified to be properly constructed and validated, they will be integrated into the specific flow of the Quantum SafeML method outlined previously. This integration will depend on the particular conditions of different quantum machine learning scenarios, and may vary accordingly. However, a general understanding of how the functions are employed within the overall method will be established at this stage.

5.5 Testing

In the testing phase, we will use the implemented quantum machine learning models and ensure that their data—both input and output—is formatted to match the required types for the quantum distance functions. From here, the distance metrics can be applied, as per the SafeML method, and their values can be evaluated to determine their effectiveness in assessing model safety.

At this stage, both model types—VQC and QCNN—must be tested for their specific use cases. Fortunately, the Qiskit example labs provide a VQC notebook that demonstrates how the model operates on the Iris dataset problem. This is compared to a classical SVM model. In this case, we will apply the classical SafeML method to the SVM, and the Quantum SafeML method to the VQC, and then compare the resulting metrics.

Similarly, the SafeML example lab for German traffic sign recognition uses a CNN for classification and applies the classical SafeML method to demonstrate the process. The quantum convolutional neural network is a direct model equivalent, and will be instantiated and trained on the same data. The results of applying the Quantum SafeML method to this QCNN will then be analyzed.

The analysis of the effectiveness of Quantum SafeML will ideally follow the approach used in classical SafeML evaluation. In the original SafeML project, evaluation is done by comparing the values of the distance measures to the minimum true accuracy of the model's predictions. A similar evaluation approach will be applied here.

Established techniques for measuring the accuracy of classifier models will be used to provide a performance baseline. This will allow us to assess how the distance metric values relate to overall model accuracy. A correlation is expected between the overall quality of model accuracy and the magnitude of the observed metric results on the model's predictions.

5.5.1 VQCs on Toy Datasets

The Iris dataset is a common toy dataset used for classifying types of flowers based on a set of features. An existing notebook in the Qiskit documentation demonstrates a direct comparison between a classical classifier and a VQC on this dataset. The Quantum SafeML method will be applied in this context.

To further evaluate VQC performance from a SafeML perspective, the same model architecture will be trained on multiple different datasets. This will allow us to observe how the results of the method vary as the classifier is exposed to different problem domains.

5.5.2 Digit Prediction Using a QCNN

The SafeML example lab for German traffic sign recognition applies the classical SafeML method to CNN classification results. As the quantum convolutional neural network is a direct model counterpart, the Quantum SafeML method should also be applied to this problem.

However, the complexity of the German traffic sign dataset exceeds the capabilities of the quantum hardware simulators used in this project. Therefore, a simpler dataset will be selected for a similar classification task.

Specifically, a toy dataset composed of hand-drawn digits ranging from 0 to 9 will be used. A QCNN will be trained to classify these digits, and the Quantum SafeML method will be applied to its predictions. Variations in the results will be analyzed across the different digit classes.

5.6 Evaluation

Once the example notebooks have been run and amendments made to apply the SafeML method to them, the resulting outputs will be evaluated.

5.6.1 Visualizing Results

Graphs will be generated to show how the distance metric values vary across sets of predictions, and how these variations relate to the true accuracy of the datasets.

Visualizations for Distance Metrics and Performance

The following visualizations will be used to assess and compare the performance of different distance metrics, as well as the accuracy of the quantum models:

- `visualization_metric_scores_vqc`
  A plot showing the metric scores (e.g., fidelity, Bures distance, trace distance, or relative entropy) across different datasets used in a Variational Quantum Classifier (VQC). This will help visualize how the distance metrics respond to different data inputs.
- `visualization_metric_closeness_accuracy_vqc`
  A graph showing the relationship between distance metric scores and the accuracy of the VQC. This visualization will help determine whether higher or lower metric values correspond with better model performance.
- `visualization_metric_performance_qcnn_classes`
  A graph illustrating how the performance of the distance metrics varies across different classes in a Quantum Convolutional Neural Network (QCNN). This will help assess whether the metrics can capture class-specific performance trends.
- `confusion_matrix_qcnn_misclassifications`
  A confusion matrix highlighting misclassifications made by the QCNN. This visualization will provide insight into which classes are most frequently confused, helping to identify areas where the model struggles.

5.6.2 Analysis of Results

The results generated by the distance metric functions will be recorded and compared to the accuracy scores of each classifier model. Commentary will be provided on how well the performance of the distance metrics aligns with overall model accuracy, and whether these metrics effectively highlight inconsistencies or weaknesses in the model's predictions.

5.7 Limitations of Methodology

Availability of Quantum Distance Metrics

One of the primary limitations of this methodology is the potential scarcity of quantum distance metrics that can be sourced and analyzed.

This scarcity may stem from the relatively recent and experimental nature of quantum technology, which has resulted in limited exploration into this specific area. To mitigate this, the search will begin with well-documented and known quantum equivalents to the metrics used in classical SafeML, ensuring at least a foundational set of metrics for analysis.

Quantum Simulation Capacity

It must be emphasized that this project is constrained by the capacity of available quantum simulation tools. While the chosen frameworks and selected model architectures aim to make the most of the resources available, this remains a necessary limitation due to the restricted access to real quantum hardware. The performance of the Quantum SafeML method, therefore, must be considered within the bounds of simulated environments.

Accuracy of Quantum Architecture Simulation

Quantum architecture can be simulated in several different ways, each with its own respective strengths and weaknesses. Regardless of the simulation method ultimately chosen, it is important to acknowledge that no simulation is expected to fully replicate the behavior of a real quantum computing system.

With this expectation in mind, any known shortcomings of the selected simulation approach will be documented, along with a discussion of how these may impact the final evaluation and deployment of the Quantum SafeML method. This will help contextualize the results and ensure transparency around the reliability of the findings.

5.8 Methodology Summary

This study focuses on adapting the SafeML method for Quantum Machine Learning by integrating quantum distance metrics.

The methodology begins with sourcing and analyzing a range of quantum distance metrics, converting their mathematical foundations into pseudo-code, and implementing them in a programming environment compatible with quantum computing.

The classical SafeML methodology will be studied and modified to support quantum classifiers, ensuring compatibility with quantum-based statistical measures. Due to the inaccessibility of real quantum hardware, the study will utilize quantum simulation frameworks such as Qiskit or PennyLane to build and evaluate Quantum Machine Learning models.

Once the quantum models are implemented, quantum distance metrics will be applied to evaluate their performance—particularly by comparing the models' predictions against actual values.

The effectiveness of Quantum SafeML will be assessed using established analysis techniques drawn from classical SafeML research, aiming to preserve consistency in evaluation.

Finally, limitations such as the availability of quantum distance metrics and the accuracy of quantum simulations will be considered, with a discussion on how these factors may influence the results.

The overall approach ensures a structured and methodical adaptation of SafeML principles to the quantum domain while accounting for current technological constraints.

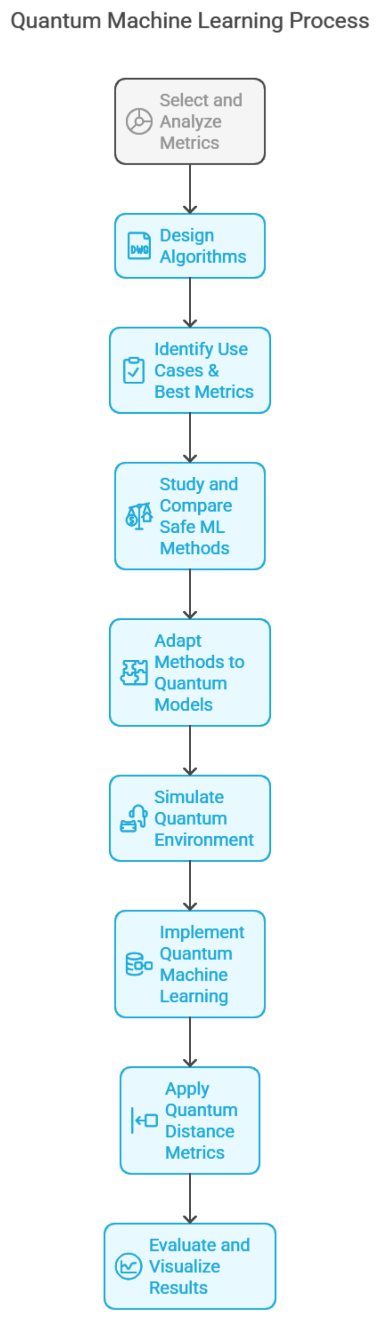

Quantum Machine Learning Process
Select and Analyze Metrics
Design Algorithms
Identify Use Cases & Best Metrics
Study and Compare Safe ML Methods
Adapt Methods to Quantum Models
Simulate Quantum Environment
Implement Quantum Machine Learning
Apply Quantum Distance Metrics
Evaluate and Visualize Results

Figure 5.2 A flowchart showing how the methodology will be undertaken

Section	**Key Points**
Quantum Distance Metrics	Quantum distance metrics will be sourced, analyzed, and translated into pseudo-code before being implemented in a quantum computing-compatible programming environment.
SafeML Method Adaptation	The classical SafeML methodology will be adapted for quantum classifiers, ensuring compatibility with quantum statistical frameworks.
Quantum Environment Simulation	Quantum simulations using Qiskit or PennyLane will replace real quantum hardware, allowing for the testing and evaluation of Quantum Machine Learning models.
Model Implementation and Evaluation	After implementation, quantum distance metrics will be applied to assess model performance, comparing predicted and actual outputs.
Assessment of Quantum SafeML	The effectiveness of the adapted SafeML approach will be evaluated using methods consistent with those in classical SafeML research.
Limitations	Key limitations include the availability of suitable quantum distance metrics and the fidelity of quantum simulation, both of which are discussed in terms of their potential impact.
Overall Approach	The study provides a structured adaptation of SafeML to the quantum domain, mindful of current technological boundaries.

Table 5.5 Summary of Quantum SafeML Methodology

Chapter 6

Experiments

6.1 Experiments Introduction

This chapter covers experimentation with the existing technologies upon which Quantum SafeML is built. This includes classical SafeML, quantum machine learning, and quantum distance metrics.

6.1.1 SafeML Example Analysis

To begin, existing example applications of SafeML will be observed and analyzed for their functionality and the underlying steps within the process. A discussion of the required changes to adapt these examples for quantum solutions will also be included.

6.1.2 Qiskit Environment Instantiation

The Qiskit environment will be instantiated, and, using this platform, research will be conducted into quantum classification models. A discussion will follow regarding how quantum machine learning differs from classical machine learning, with particular focus on how these differences impact classifier-based applications specifically.

6.1.3 Quantum Machine Learning Model Construction

Specific model architectures will be explored, and decisions will be made regarding which architectures to implement for further experimentation. These decisions will be based on how well the models can be instantiated within the chosen quantum computing simulation environment, as well as their compatibility with the ideal use cases for the selected quantum distance metrics.

6.1.4 Quantum Distance Metrics

Finally, a range of quantum distance metrics will be sourced and analyzed, ultimately being designed in translatable formats suitable for potential implementation—likely in the form of pseudo-code. A comparison of the use cases for each of the proposed metrics will then be carried out, evaluating how they may best suit particular quantum machine learning problems. With these considerations in mind, a verdict will be reached to select a subset of the analyzed metrics for further development into applicable functions for use on quantum classifiers.

6.2 SafeML Example Implementation - German Traffic Sign Classification

An example of SafeML is provided, demonstrating its application on an existing classifier solution that operates on the German traffic sign dataset in order to predict new images and their corresponding meanings in a road context.

The example discussed here can be found at SafeML GTSRB CNN Implementation.

Importing Data

To begin, the existing German traffic sign dataset is imported into the workspace. This data will be used to train, test, and evaluate the implemented classifier.

To add robustness, the dataset is then shuffled using a consistent random seed.

Splitting Data for Train, Test, and Validation

The dataset is split, saving twenty percent of the data for validation and testing, while the remaining eighty percent is used for the training phase. The labels are also converted with one-hot encoding at this stage to ensure that the categorical data of the classes are represented numerically.

Defining the CNN Model and Its Architecture

The Convolutional Neural Network's (CNN) structure is defined here, dictating how it is built and how it processes the input data.

Algorithm 1 CNN Model for Image Classification

```
1: **Initialize** the model as a Sequential
2: **Add** Conv2D layer with 32 filters, kernel size (5,5), activation function ReLU, and input
   shape ($X_{train}.shape[1:]$)
3: **Add** Conv2D layer with 64 filters, kernel size (3,3), and activation function ReLU
4: **Add** MaxPool2D layer with pool size (2,2)
5: **Add** Dropout layer with dropout rate 0.25
6: **Add** Conv2D layer with 64 filters, kernel size (3,3), and activation function ReLU
7: **Add** MaxPool2D layer with pool size (2,2)
8: **Add** Dropout layer with dropout rate 0.25
9: **Flatten** the output
10: **Add** Dense layer with 256 units and activation function ReLU
11: **Add** Dropout layer with dropout rate 0.5
12: **Add** Dense layer with 43 units and activation function softmax
13: **Compile** the model with:
14:   loss='categorical_crossentropy'
15:   optimizer='adam'
16:   metrics=['accuracy']
```

The first layer is a 2D convolutional layer (`Conv2D`) with 32 filters, each of size (5, 5). This layer applies convolution operations to the input image, detecting low-level features such as edges and textures. The activation function used here is ReLU, which introduces non-linearity into the model, allowing it to learn more complex patterns.

Next, another 2D convolutional layer is added with 64 filters of size (3, 3) and the same ReLU activation function. This layer captures more complex features, such as shapes or object parts. The smaller filter size (3, 3) enables the model to focus on finer details of the image. Increasing the number of filters to 64 allows the network to capture a wider variety of features from the image.

Following the second convolutional layer, a max pooling layer (`MaxPool2D`) is added with a pool size of (2, 2). This operation reduces the spatial dimensions of the feature map by selecting the maximum value from each 2x2 patch, effectively down-sampling the input while retaining important features. Pooling also reduces the computational complexity of the model and helps prevent overfitting by making the model more invariant to small translations in the input image.

A dropout layer with a rate of 0.25 follows the pooling layer. Dropout is a regularization technique that randomly sets 25% of the input units to zero during training. This prevents the network from becoming too reliant on any individual neuron and helps mitigate overfitting.

A third convolutional layer, similar to the second one, is added with 64 filters of size (3, 3), again using the ReLU activation function. This layer captures even more complex patterns and higher-level features. After this, another max pooling layer is applied to further down-sample the output.

Another dropout layer with a rate of 0.25 follows the third pooling layer to continue the regularization process, further ensuring the model doesn't overfit.

The output of the convolutional and pooling layers is flattened into a one-dimensional vector using the `Flatten` layer. This prepares the data for the fully connected layers that follow.

A fully connected (dense) layer with 256 units and ReLU activation follows the flattening operation. This layer allows the model to integrate the high-level features extracted by the convolutional layers. The ReLU activation function again introduces non-linearity, helping the model to learn complex relationships in the data.

Another dropout layer, this time with a rate of 0.5, is added after the fully connected layer. This higher dropout rate helps to prevent overfitting, especially in the fully connected layers, which typically contain a large number of parameters.

Finally, the output layer is a dense layer with 43 units, corresponding to the number of classes in the classification task. The softmax activation function is used in the output layer to output class probabilities, which ensures the sum of all output values is equal to 1, making it suitable for multi-class classification tasks.

Training the Model and Calculating Its Accuracy

Here, the model is fitted to the training data, and a series of epochs iteratively adjusts the weights between layers to increase accuracy and decrease loss.

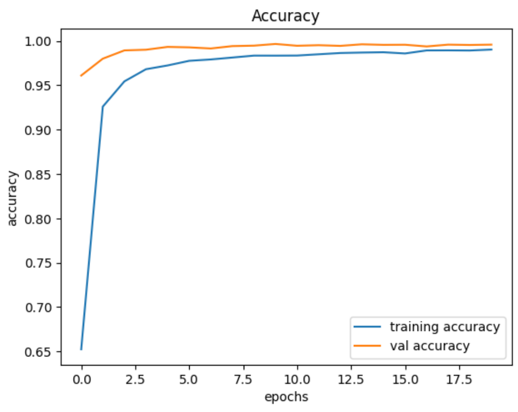


Figure 6.1 Accuracy graph for the CNN on German traffic sign data

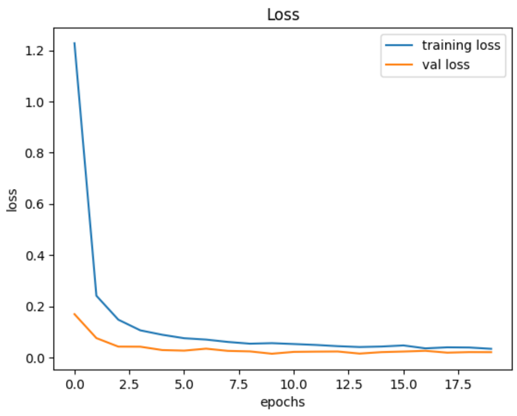


Figure 6.2 Loss graph for the CNN on German traffic sign data

Applying the Model on Test Data

The model is then fed the data previously allocated for testing, yielding impressive prediction results.

Metric	**Value**
Accuracy Score	0.9680

Table 6.1 Accuracy score of the CNN classifier on the German Traffic Sign dataset

This is a high score, but not perfect, meaning it is the job of SafeML to locate discrepancies within the predictions.

Comparing the True Labels with Predicted Labels and Using Statistical Parametric Mapping as the SafeML Method

At this stage, a class is chosen for analysis, and the dataset is traversed to find all datapoints that belong to this class. These form a set of trusted datapoints. Following this, the prediction set is traversed, and all datapoints incorrectly labeled into the chosen class will be extracted, forming an untrusted dataset.

These two datasets are compared using statistical parametric mapping.

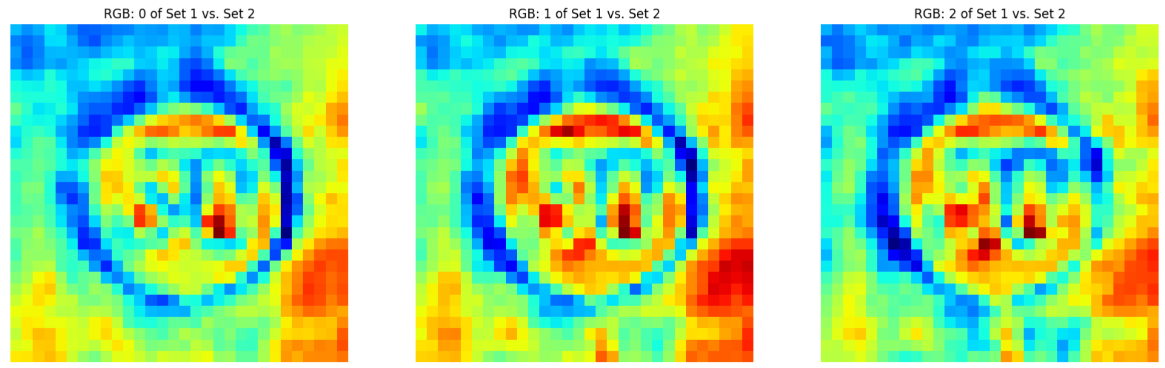


Figure 6.3 Heatmap of confidence on image predictions by pixel.

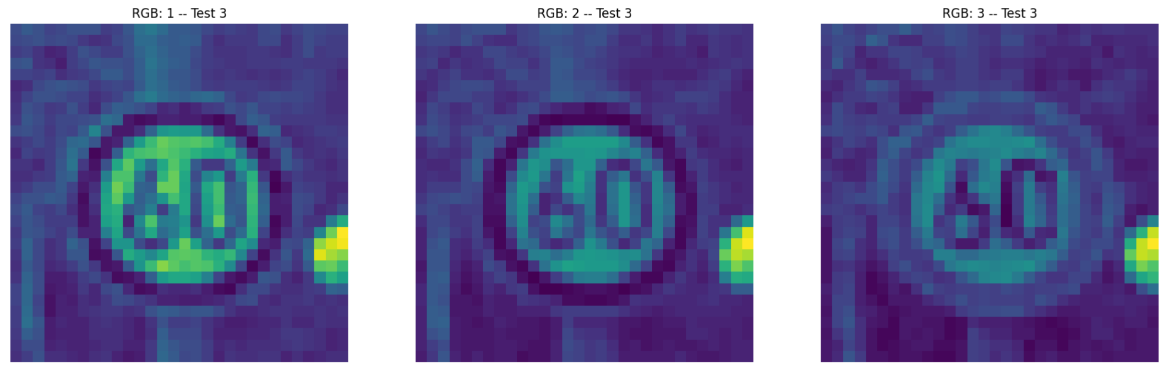


Figure 6.4 Correctly classified images.

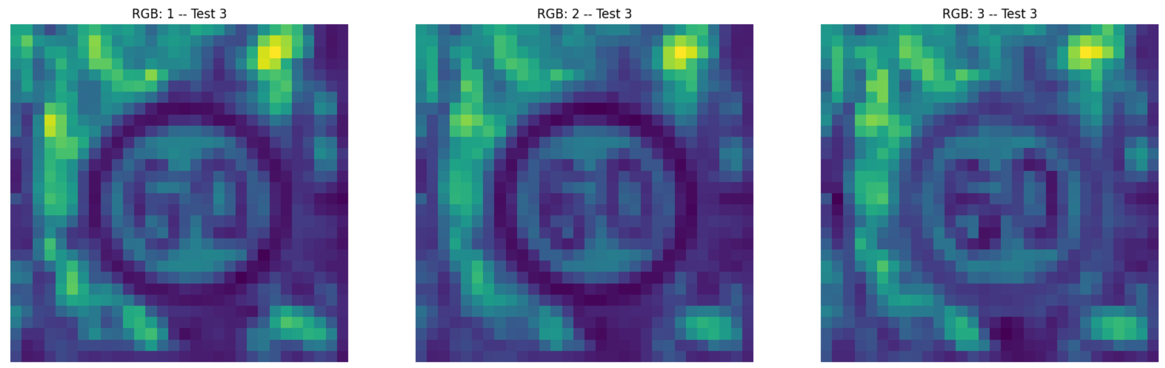


Figure 6.5 Incorrectly classified images.

Output Value	Interpretation
1.0002	The model's misclassified samples are highly different from the correctly classified training samples. This suggests a clear distinction in features.
0.7626	Moderate difference—misclassified samples are still quite distinct but not as much as the first case.
0.6410	Misclassified samples are somewhat similar to the correctly classified ones. This means the model is making mistakes on hard-to-distinguish cases.

Table 6.2 Interpretation of statistical differences between misclassified and correctly classified samples.

In the above figures, it can be observed that discrepancies arise in images that are particularly difficult for the classifier model.

Safety Monitoring through Statistical Parametric Mapping

The mapping is operated, and the following outputs are given.

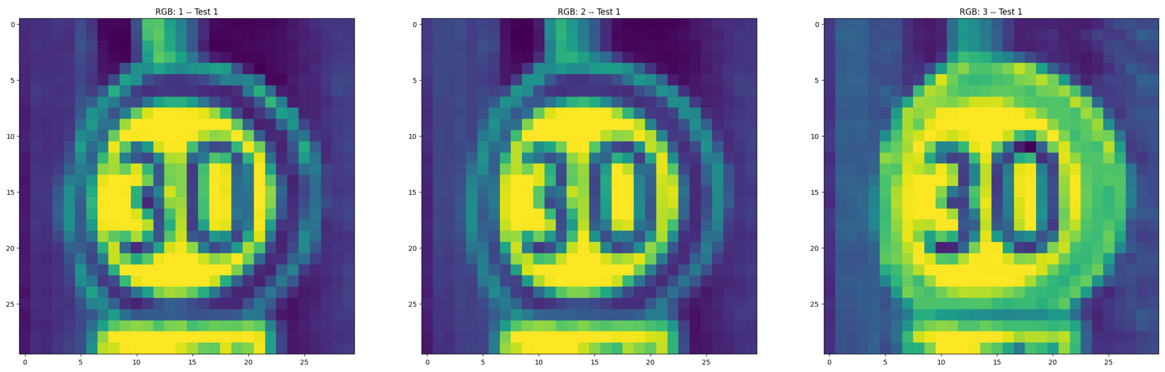


Figure 6.6 RGB color channels of a test image.

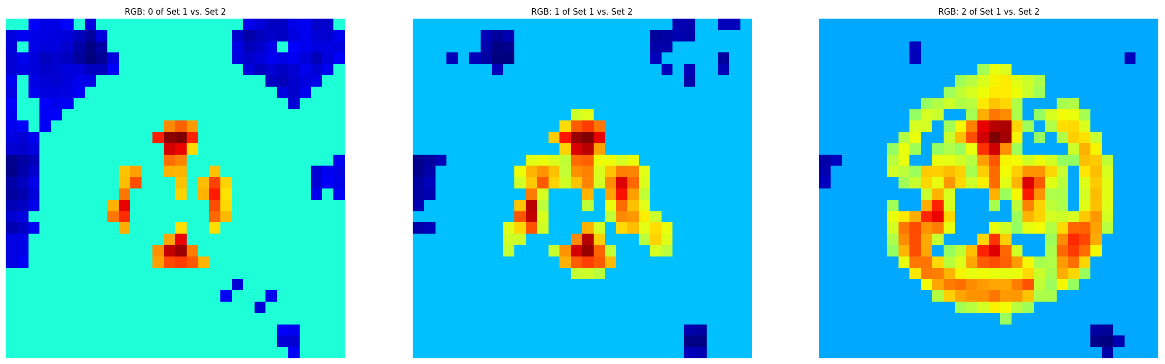


Figure 6.7 Heatmaps of statistical differences between correctly and incorrectly classified samples.

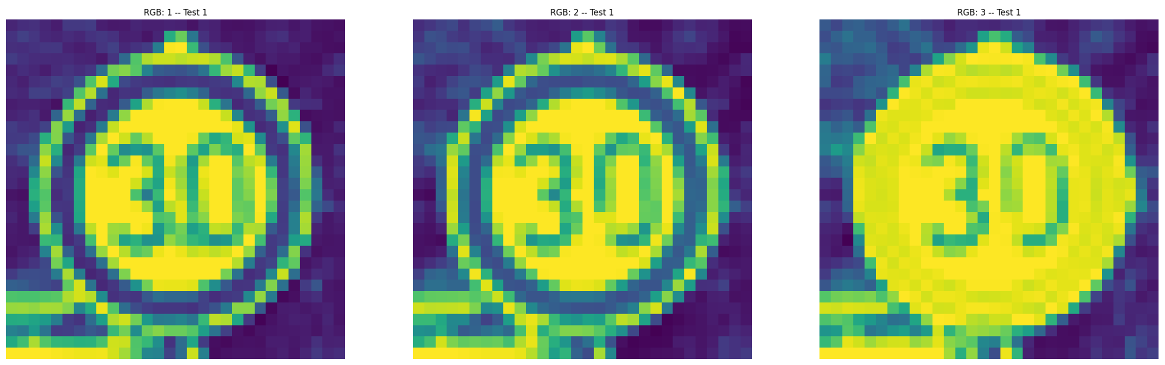


Figure 6.8 Reference images of correctly classified test samples.

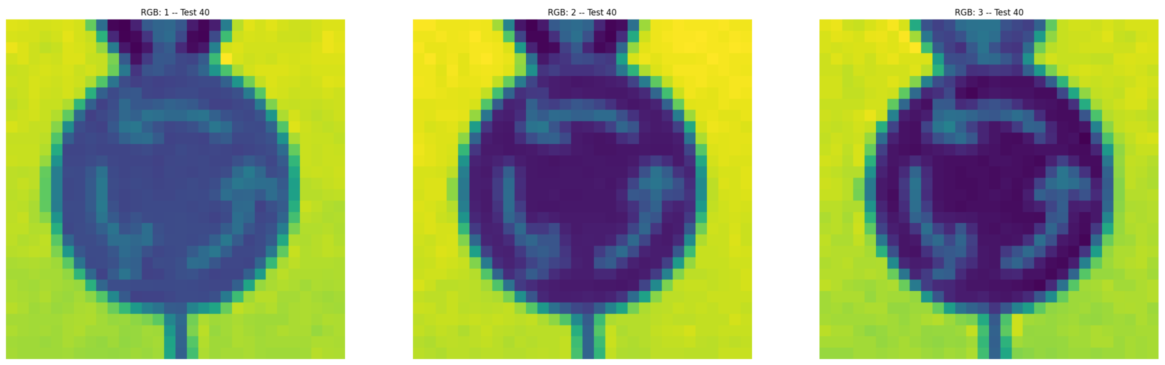


Figure 6.9 Reference images of misclassified test samples.

6.2.1 Analysis

Overall, the SafeML method serves as an effective and interpretable way of providing an added layer of confidence monitoring for classifier systems.

Key Characteristics

The primary aspects of the SafeML method are observed to be an adequately trained classifier, which comprises most of the offline training phase. This is of importance in real-world systems, as it ensures that the overall system is operating effectively. SafeML can catch and report scenarios where classifiers under-perform in terms of confidence, but it does not itself provide an alternate system if the current one is regularly ineffective for the problem the solution is built for.

The next key aspect is a buffer that allows the classifier to be accessed by the larger system when appropriately commanded. However, an important part of this step is a check to evaluate whether the data the classifier will be passed is of an acceptable size and shape. This prevents potential command failures of the classifier by ensuring adequate input for predictions to be made.

After this, the classifier behaves as constructed and makes predictions based on the data it is passed by the system. These predictions are then used to determine the values of the statistical measures SafeML specifies at runtime.

A function is employed on the chosen classifier implementation to compute the objective true accuracy of the model, and this is compared to the verdicts of the distance measures.

A check is used on the outcome of this comparison, and this ultimately determines the next steps of the system.

These steps form the basis of the SafeML method, which, in its abstract form, should be widely applicable to most classifier solutions. Tweaking may be required to extend it further to quantum domains, particularly through the use of quantum distance metrics, which are explored later in this chapter.

6.3 Instantiating Qiskit

Qiskit is used to instantiate the model environment. Recently, Qiskit underwent a major update, which reorganized many aspects of the library.

Although documentation on this new migration is currently limited, the correct methods for using the new functionalities were deduced in order to set up a simulation of a qubit on a classical machine.

The basic structure of a quantum simulator involves creating a quantum circuit that maps simulated qubits to representations using classical bits. Quantum gates are then applied to control the logical flow of the system. The qubits can be measured in order to visualize their states, and the simulator can be run using a backend. The simulator will then map values to either the states 1 or 0.

Algorithm 2 Simulate a Quantum Circuit using Qiskit

```
Require: A quantum circuit $QC$ with $n$ qubits and $m$ classical bits
Ensure: Measurement results of the quantum circuit
  Create a quantum circuit $qc$ with $n$ qubits and $m$ classical bits
  Apply Hadamard gate on qubit 0: $H(0)$
  Apply CNOT gate with control qubit 0 and target qubit 1: $CX(0,1)$
  Measure the qubits into classical bits: $Measure([0,1],[0,1])$
  Choose the backend simulator:
      $simulator \leftarrow Aer.get\_backend(”qasm\_simulator”)$
  Run the quantum circuit on the simulator with 1024 shots:
      $job \leftarrow simulator.run(qc, shots = 1024)$
  Retrieve the results of the simulation:
      $result \leftarrow job.result()$
  Get the measurement results:
      $counts \leftarrow result.get\_counts(qc)$
  Print the measurement results: $Print(counts)$
  Visualize the measurement results in a histogram: $plot\_histogram(counts)$
  return $counts$
```

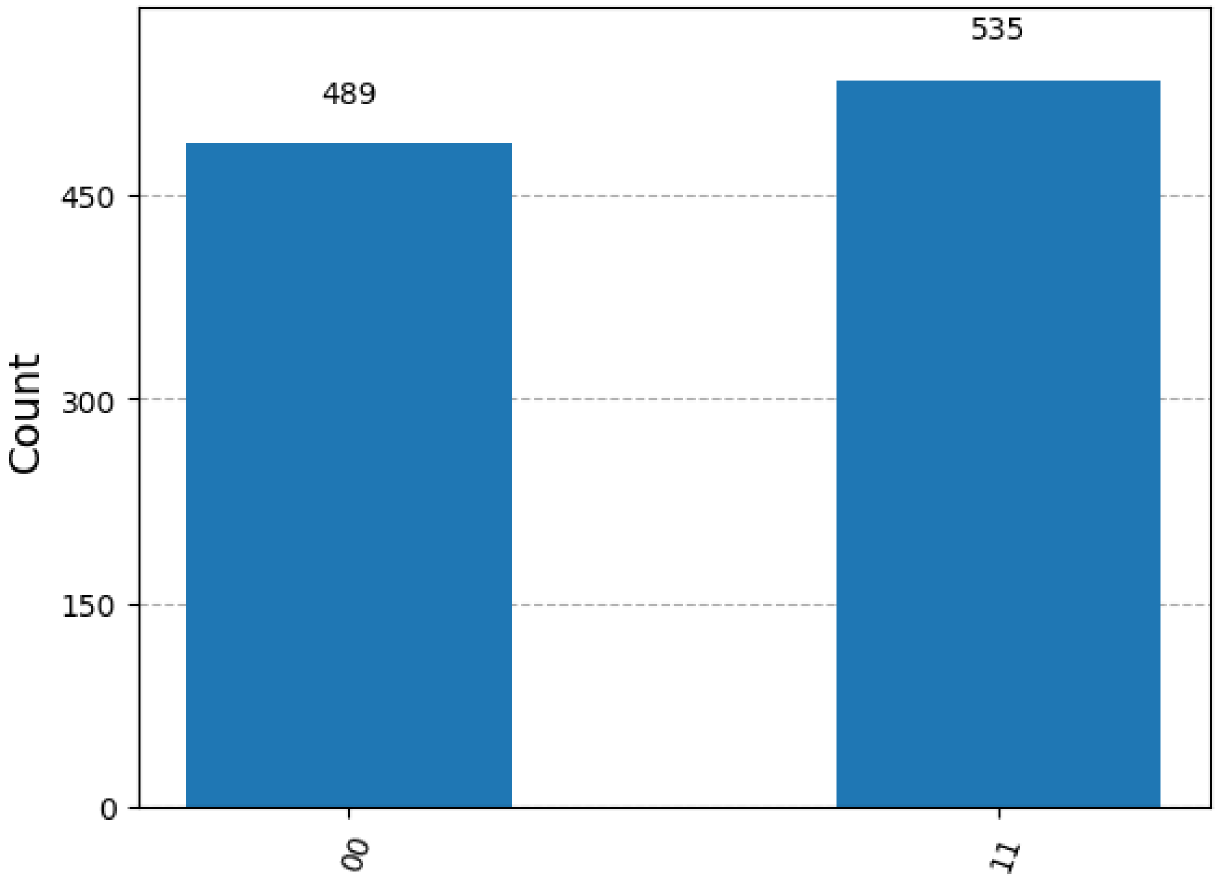


Figure 6.10 Output graph of a basic quantum circuit simulation.

6.3.1 Setting up Qiskit for Machine Learning

While the setup process may vary from model to model, Qiskit's machine learning functionality is versatile and can be applied to many quantum machine learning solutions.

Qiskit's machine learning capabilities include integration with PyTorch, feature maps, kernel-based methods, as well as Quantum Neural Networks (QNN). These features will allow for a more streamlined production of QNN models and provide an extension to Quantum Convolutional Neural Networks (QCNN) models, which are aimed to be implemented during the development phase of this project.

6.4 Researching Quantum Distance Metrics

6.4.1 Quantum Distance Measures

SafeML, as implemented in classical systems, relies on conventional distance metrics to determine the difference between predicted and actual values in a dataset. This approximation helps estimate the model's confidence in its predictions. To extend the SafeML approach to quantum systems, it is essential to explore various quantum distance metrics to determine which are suitable for a quantum SafeML implementation. These metrics typically compare density matrices representing probability distributions to quantify their overlap.

Trace Distance

Trace distance measures the distinguishability between two quantum states, offering a probabilistic measure of correctly identifying which state is which.

The trace distance between two density matrices ρ and σ is defined as:

$$D(\rho, \sigma) = \frac{1}{2} \mathrm{Tr} \, |\rho - \sigma| \tag{6.1}$$

Here, $|\rho - \sigma|$ denotes the absolute value (or trace norm) of the matrix. $D = 0$ indicates that the states are identical, while $D = 1$ implies completely distinguishable states.

Algorithm 3 Compute the Trace Distance Between Two Probability Distributions

```
Require: Two probability distributions $P_1, P_2$
Ensure: Trace distance $T$ between $P_1$ and $P_2$
  Convert $P_1, P_2$ into array form
  if shape of $P_1 \neq$ shape of $P_2$ then
      Raise error: "Distributions must have the same shape."
  end if
  $D \leftarrow |P_1 - P_2|$
  $S \leftarrow \sum D$
  $T \leftarrow \frac{1}{2}S$
  return $T$
```

Fidelity

Fidelity quantifies the overlap between two quantum states, where $F = 1$ indicates identical states and $F = 0$ indicates no overlap.

$$F(\rho, \sigma) = \left(\mathrm{Tr} \sqrt{\sqrt{\rho}\, \sigma \sqrt{\rho}} \right)^2 \tag{6.2}$$

Algorithm 4 Compute the Fidelity Between Two Density Matrices

```
Require: Density matrices $\rho_1, \rho_2$
Ensure: Fidelity $F$
  Convert $\rho_1, \rho_2$ to matrix representations if necessary
  if shapes differ then
      Raise error: "Matrices must have the same shape."
  end if
  sqrt_rho1 $\leftarrow$ Cholesky$(\rho_1 + \varepsilon I)$
  intermediate $\leftarrow$ sqrt_rho1 $\cdot \rho_2 \cdot$ sqrt_rho1
  sqrt_intermediate $\leftarrow$ MatrixSqrt(intermediate)
  $F \leftarrow$ (Re(Tr(sqrt_intermediate)))$^2$
  return $F$
```

Bures Distance

The Bures distance is derived from fidelity and provides a metric for the geometric distance between quantum states:

$$D_B(\rho, \sigma) = \sqrt{2\left(1 - \sqrt{F(\rho, \sigma)}\right)} \tag{6.3}$$

Algorithm 5 Compute the Bures Distance Between Two Density Matrices

```
Require: Density matrices $\rho_1$, $\rho_2$
Ensure: Bures distance $D_B$
  Convert $\rho_1$, $\rho_2$ to matrix representations if needed
  if shapes differ then
      Raise error: "Matrices must have the same shape."
  end if
  sqrt_rho1 $\leftarrow$ Cholesky$(\rho_1 + \varepsilon I)$
  intermediate $\leftarrow$ sqrt_rho1 $\cdot\, \rho_2 \cdot$ sqrt_rho1
  sqrt_intermediate $\leftarrow$ MatrixSqrt(intermediate)
  $F \leftarrow (\mathrm{Re}(\mathrm{Tr}(\text{sqrt\_intermediate})))^2$
  $D_B \leftarrow \sqrt{2(1-\sqrt{F})}$
  return $D_B$
```

Quantum Relative Entropy

Quantum relative entropy quantifies the information loss when approximating ρ with σ:

$$S(\rho\|\sigma) = \mathrm{Tr}\left(\rho \log \rho - \rho \log \sigma\right) \tag{6.4}$$

Algorithm 6 Compute the Quantum Relative Entropy

```
Require: Density matrices $\rho_1$, $\rho_2$
Ensure: Quantum relative entropy $S(\rho_1\|\rho_2)$
  Ensure both matrices are valid density matrices (shape, trace 1, PSD)
  Perform eigendecomposition:
      $\rho_1 = V_1 \Lambda_1 V_1^\dagger, \quad \rho_2 = V_2 \Lambda_2 V_2^\dagger$
  Compute logarithms:
      $\log \rho_1 = V_1 \log(\Lambda_1 + \varepsilon) V_1^\dagger, \quad \log \rho_2 = V_2 \log(\Lambda_2 + \varepsilon) V_2^\dagger$
  $S \leftarrow \mathrm{Re}(\mathrm{Tr}(\rho_1(\log \rho_1 - \log \rho_2)))$
  return $S$
```

Hellinger Distance

The Hellinger distance provides a fidelity-based distance metric:

$$H(\rho,\sigma) = \sqrt{1-\sqrt{F(\rho,\sigma)}} \tag{6.5}$$

Algorithm 7 Compute the Hellinger Distance Between Density Matrices

```
Require: Density matrices $\rho_1$, $\rho_2$
Ensure: Hellinger distance $H$
  Convert and validate shapes
  Compute sqrt_rho1 $\leftarrow$ Cholesky$(\rho_1 + \varepsilon I)$
  intermediate $\leftarrow$ sqrt_rho1 $\cdot \rho_2 \cdot$ sqrt_rho1
  sqrt_intermediate $\leftarrow$ MatrixSqrt(intermediate)
  trace_val $\leftarrow$ Re(Tr(sqrt_intermediate))
  $H \leftarrow \sqrt{1 - \text{trace\_val}}$
  return $H$
```

Quantum Wasserstein Distance (Approximate)

An approximate form of quantum Wasserstein distance uses the Frobenius norm:

$$D_W(\rho, \sigma) \approx \|\rho - \sigma\|_F \tag{6.6}$$

Algorithm 8 Compute the Approximate Quantum Wasserstein Distance

```
Require: Density matrices $\rho_1$, $\rho_2$
Ensure: Wasserstein distance $D_W$
  Ensure valid density matrices
  Compute Frobenius norm:
                    $D_W \leftarrow \|\rho_1 - \rho_2\|_F$
  return $D_W$
```

Euclidean Distance (Classical Baseline)

The classical Euclidean distance is given by:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_i (x_i - y_i)^2} \tag{6.7}$$

Quantum Jensen-Shannon Divergence

Defined using quantum relative entropy:

$$QJSD(\rho, \sigma) = \frac{1}{2} S\left(\rho \,\|\, \frac{\rho + \sigma}{2}\right) + \frac{1}{2} S\left(\sigma \,\|\, \frac{\rho + \sigma}{2}\right) \tag{6.8}$$

Algorithm 9 Compute the Euclidean Distance Between Vectors

```
Require: Vectors $\vec{v}_1$, $\vec{v}_2$
Ensure: Euclidean distance $d$
  if shape mismatch then
      Raise error: "Vectors must be the same shape."
  end if
  $d \leftarrow \|\vec{v}_1 - \vec{v}_2\|$
  return $d$
```

Algorithm 10 Compute the Quantum Jensen-Shannon Divergence (QJSD)

```
Require: Density matrices $\rho$, $\sigma$
Ensure: QJSD value
  Compute the average state: $\mu \leftarrow \frac{1}{2}(\rho + \sigma)$
  Compute eigen-decompositions and matrix logarithms:
      $$\log\rho, \log\sigma, \log\mu \text{ (add } \varepsilon I \text{ as needed for stability)}$$
  Compute quantum relative entropy terms:
      $$S_1 \leftarrow \mathrm{Tr}(\rho(\log\rho - \log\mu))$$
      $$S_2 \leftarrow \mathrm{Tr}(\sigma(\log\sigma - \log\mu))$$
  $QJSD \leftarrow \frac{1}{2}(S_1 + S_2)$
  return $QJSD$
```

6.5 Applying Quantum Distance Metrics to the Safe ML Structure

In exploring quantum distance metrics, we analyze the main candidate formulas used within quantum systems.

It is important to note that the key differences between these metrics may make each optimal for different types of systems.

As in classical safe machine learning, each metric naturally lends itself to identifying discrepancies between two datasets. However, in the early stages of quantum machine learning implementation, some metrics will be better suited for simpler model architectures and smaller, illustrative datasets.

Some potentially useful metrics include fidelity, quantum relative entropy, trace distance, and Hellinger distance.

Fidelity is useful as a baseline for evaluating the general overlap between two quantum states or distributions. In an era where technologies such as ChatGPT and DeepSeek have set a precedent for the proliferation of generative AI models, there is growing demand for reliable ways to determine whether data has been generated by such models. In the context of quantum machine learning, fidelity can be applied to examine how easily one quantum state (e.g., original data) could be mistaken for another (e.g., generated data). Since fidelity measures the similarity between quantum states, it could be incorporated into the discriminator section of a qGAN model to evaluate the model's ability to distinguish between generated and authentic samples.

Quantum Relative Entropy has important use cases as well, as it is analogous to the classical Kullback-Leibler divergence often used in training neural networks. Due to this connection, quantum relative entropy is particularly valuable for determining how distinguishable two quantum states are. This can be used to evaluate how the predicted quantum states from a quantum neural network (QNN) differ from the true or target states.

Trace Distance may be especially useful in the safety monitoring of quantum machine learning systems. It also measures the distinguishability between two quantum states and can be used to compare density matrices, making it ideal for evaluating the difference between the pure state output of a quantum classifier and the true target values. Given the current reliance on quantum simulation—due to limited access to quantum hardware—trace distance provides a means of detecting subtle differences in probability distributions. This is crucial for addressing risks such as quantum noise, which frequently reduces the reliability and accuracy of quantum models. It can also be employed in binary classification problems to facilitate interpretation of quantum data using classical systems.

Bures Distance is another valuable metric for understanding model performance, especially in noisy environments. By taking quantum noise into account, it can help assess the variation and robustness of the model's predictions. Given the sensitive conditions required for quantum computation, having a method to evaluate how noise affects overall model performance is important.

A summary of potential use cases for each metric is presented in the table below:

Metric	**Best For**	**Example Use Case**
Trace Distance	Distinguishability, error bounds, binary classification	Comparing two pure states in a quantum classifier.
Fidelity	Overlap, probabilistic models, training monitoring	Evaluating qGAN performance.
Bures Distance	Mixed states, robustness, geometric insights	Measuring robustness in noisy quantum classifiers.
Quantum Relative Entropy	Discrepancy analysis, interpretability, calibration	Identifying unsafe predictions in quantum neural networks.
Hellinger Distance	Probability distribution comparisons, small-scale differences	Comparing predicted and true quantum reward distributions.
Quantum Wasserstein	Resource allocation, noise sensitivity, higher-level interpretability	Measuring transport cost in generative quantum models.
Hilbert-Schmidt Distance	Computational efficiency, global differences, large-scale systems	Screening discrepancies in large quantum datasets.

Table 6.3 Comparison of Various Quantum Metrics and Their Use Cases

6.6 Experiments Summary

This chapter presented experimental analyses of quantum distance metrics, the SafeML methodology, quantum simulations, and quantum machine learning methods. The results highlight key insights into both the capabilities and challenges of quantum computing in the context of safe machine learning.

6.6.1 SafeML Analysis

An example traversal of the German Traffic Sign SafeML notebook provided insight into how SafeML operates within classical systems. This offered a clear foundation upon which quantum adaptations can be developed.

6.6.2 Qiskit Environment

The Qiskit framework enables streamlined instantiation of quantum computing architectures, including the construction of quantum circuits and their associated qubits. It serves as a foundational tool for designing and simulating quantum algorithms.

6.6.3 QML Models

Example notebooks for Variational Quantum Classifiers (VQC) and Quantum Convolutional Neural Networks (QCNN) demonstrated how quantum machine learning models can be implemented and trained using Qiskit. These examples strengthen the foundation for integrating SafeML principles into hybrid quantum-classical model architectures.

6.6.4 Quantum Distance Metrics

A wide range of quantum distance metrics were sourced, compiled, and analyzed with respect to their potential applications. Each metric was assessed for its suitability in specific quantum machine learning contexts, forming a preliminary framework for safe model evaluation in quantum systems.

Chapter 7

Development

7.1 Development Introduction

In this chapter, the construction of the framework for the Quantum SafeML method is detailed. This involves converting the initial designs for quantum distance metrics into reusable and testable code modules.

Following this, hybrid quantum machine learning models will be instantiated and validated using test datasets. These basic model architectures will serve as foundational examples for evaluating the Quantum SafeML methodology.

Finally, the proposed quantum distance metric functions will be applied and tested for compatibility with the hybrid models. Successful validation indicates readiness for subsequent integration and evaluation within the full SafeML pipeline.

7.2 Building Distance Metric Functions

The quantum distance metrics—including Quantum Relative Entropy, Trace Distance, Bures Distance, and Fidelity—were implemented in Python with attention to numerical stability and validation. The primary consideration was ensuring that the input density matrices were of the same shape, a prerequisite for valid metric calculations.

7.2.1 Required Libraries

Several libraries were selected to support efficient implementation, with an emphasis on numerical processing and compatibility with quantum state representations.

NumPy

NumPy provides a robust toolkit for handling multi-dimensional arrays and matrices. Its mathematical functions are essential for reshaping and processing data used in quantum simulations.

DensityMatrix

The `DensityMatrix` class from Qiskit's `quantum_info` module offers a convenient and quantum-native way to represent and manipulate quantum states. This class simplifies operations like partial traces and matrix comparisons, aligning well with our SafeML quantum extensions.

sqrtm

The `sqrtm` function from `scipy.linalg` computes the matrix square root and is required for calculating both the Bures distance and fidelity metrics, which rely on intermediate matrix root operations.

7.2.2 Fidelity

Algorithm 11 Quantum Fidelity Implemented Function

```
Require: Two density matrices $\rho_1, \rho_2$
Ensure: Fidelity $F(\rho_1, \rho_2)$
 1: Convert $\rho_1$ and $\rho_2$ to float64 for numerical precision.
 2: Compute traces: $\mathrm{Tr}(\rho_1)$ and $\mathrm{Tr}(\rho_2)$.
 3: if either trace $\approx 0$ then
 4:     Raise Error: Invalid input due to zero-trace matrix.
 5: end if
 6: Normalize both matrices.
 7: Regularize by ensuring non-negativity of all elements.
 8: Compute $\sqrt{\rho_1}$ using sqrtm with noise term $\varepsilon = 10^{-10}$.
 9: Compute $M = \sqrt{\rho_1}\rho_2\sqrt{\rho_1}$.
10: Compute $\sqrt{M}$ and calculate fidelity: $F = (\mathrm{Tr}(\sqrt{M}))^2$.
11: return $F$
```

7.2.3 Bures Distance

Algorithm 12 Bures Distance Implemented Function

```
Require: Two density matrices $\rho_1, \rho_2$
Ensure: Bures distance $D_B(\rho_1, \rho_2)$
1: Convert DensityMatrix objects to raw arrays if necessary.
2: Validate matrix shape consistency.
3: Compute $\sqrt{\rho_1}$ using Cholesky decomposition with $\varepsilon = 10^{-10}$.
4: Compute $M = \sqrt{\rho_1}\rho_2\sqrt{\rho_1}$.
5: Compute $\sqrt{M}$.
6: Compute fidelity: $F = (\mathrm{Tr}(\sqrt{M}))^2$.
7: Compute Bures distance: $D_B = \sqrt{2(1 - \sqrt{F})}$.
8: return $D_B$
```

7.2.4 Trace Distance

Algorithm 13 Trace Distance Calculation

```
Require: Two density matrices $\rho, \sigma$
Ensure: Trace distance $D_T(\rho, \sigma)$
1: Convert inputs from DensityMatrix if needed.
2: Ensure both matrices are of equal shape.
3: Compute $\Delta = \rho - \sigma$.
4: Compute trace distance: $D_T = \frac{1}{2}\sum|\Delta|$.
5: return $D_T$
```

7.2.5 Quantum Relative Entropy

Algorithm 14 Quantum Relative Entropy Implemented Function

```
Require: Two density matrices $\rho_1, \rho_2$
Ensure: $S(\rho_1||\rho_2)$
1: Convert DensityMatrix to arrays if needed.
2: Validate equal shapes, trace 1, and positive semi-definiteness.
3: Compute eigendecompositions of both matrices.
4: Apply $\log(\rho + \varepsilon)$ via eigenvalue transformation.
5: Compute:
       $S(\rho_1||\rho_2) = \mathrm{Tr}(\rho_1(\log\rho_1 - \log\rho_2))$
6: return $S$
```

7.2.6 Validation Measures

Additional safeguards were implemented to ensure that input matrices are valid:

- Matrices must be square and of equal dimensions.
- Traces must equal 1 (or be normalized).
- Matrices must be positive semidefinite.

These constraints ensure meaningful and computable metric values, avoiding runtime errors or unphysical outputs.

7.2.7 Distance Metric Unit Tests

Unit tests were created to validate the correctness and robustness of all metric functions. Each function was subjected to tests covering:

- Edge cases (e.g., identical inputs)
- Matrix shape mismatches
- Range checks (e.g., fidelity $\in [0,1]$)
- Compatibility with large and random matrices
- Symmetry (where required)

General Tests

- `test_identical_matrices`
- `test_mismatched_shapes`

Fidelity Tests

- `test_fidelity_range`
- `test_fidelity_mismatched_shapes`
- `test_fidelity_large_matrices`

Bures Distance Tests

- `test_bures_range`
- `test_bures_large_matrices`

Trace Distance Tests

- `test_trace_distance_symmetric`
- `test_trace_distance_range`
- `test_trace_distance_qiskit_density_matrix`

Quantum Relative Entropy Tests

- `test_different_matrices`
- `test_trace_mismatch`
- `test_negative_eigenvalues`
- `test_large_matrices`

7.2.8 Unit Test Results

Test	Description	Result
Distance Functions		
`test_identical_matrices`	Distance is zero for identical matrices.	Passed
`test_mismatched_shapes`	Raises error for mismatched shapes.	Passed
Fidelity		
`test_fidelity_range`	Fidelity in [0, 1] range.	Passed
`test_fidelity_mismatched_shapes`	Raises error for shape mismatch.	Passed
`test_fidelity_large_matrices`	Works with large matrices.	Passed
Bures Distance		
`test_bures_range`	Value within [0, 1] range.	Passed
`test_bures_large_matrices`	Works with large matrices.	Passed
Trace Distance		
`test_trace_distance_symmetric`	$D(A,B) = D(B,A)$.	Passed
`test_trace_distance_range`	Value within [0, 1] range.	Passed
`test_trace_distance_density_matrix`	Works with Qiskit objects.	Passed
Quantum Relative Entropy		
`test_different_matrices`	Positive entropy for different inputs.	Passed
`test_trace_mismatch`	Raises error if trace $\neq 1$.	Passed
`test_negative_eigenvalues`	Raises error for invalid matrices.	Passed
`test_large_matrices`	Handles large matrices.	Passed

Table 7.1 Unit Test Results

The unit tests verified that the implemented functions are operational and robust. The validation measures performed as expected, ensuring that the distance metrics can be applied safely and reliably across a variety of use cases. This provides a strong foundation for integrating these components into the broader Quantum SafeML methodology.

7.3 Reusable Machine learning Models

having explored example applications of both quantum machine learning model architectures, an attempt will now be made to construct re-usable functions which can call instantiation, training, and validation of these models for a streamlined QML approach.

This is good practice, not only as it will allow for more efficient testing in this project on multiple datasets, but also has implications for further development of this methodology. For instance, if deployed to a python package, example QML models could be included so basic Quantum safe Ml exercises can be reliably traversed.

Variational Quantum classifier

Constructing a variational Quantum classifier in qiskit entails standard machine learning procedures such as splitting the dataset for training testing and validation.

Once this basic reprocessing is done, a feature must be made, or rather instantiated using qiskit's circuit library, which can be used to encode the classical data into a quantum state.

Following this, a parametrized circuit is used which will form the basis of how predictions are computed.

An optimization algorithm will be chosen, here Cobyla, which will be method by which the loss is minimized by on the circuit.

A sampler is then imported.

A callback function must be specified in a VQC. Specifically in Qiskit, the VQC type takes an attribute of callback graph, which must be specified. That is done here:

Algorithm 15 Callback Graph for Plotting Objective Function Evaluation

```
Require: A list of objective function values objective_func_vals and the current objective
    function evaluation obj_func_eval
Ensure: A plot of objective function values against iterations
    Clear the output screen: clear_output(wait=True)
    Append obj_func_eval to objective_func_vals
    Set the title of the plot:
                    plt.title("Objective function value against iteration")
    Set the x-axis label:
                              plt.xlabel("Iteration")
    Set the y-axis label:
                         plt.ylabel("Objective function value")
    Plot the objective function values against iterations:
                 plt.plot(range(len(objective_func_vals)), objective_func_vals)
    Display the plot:
                                  plt.show()
```

It is worth noting that during research here, it was investigated whether a custom algorithm could be used for the callback function, as opposed to the basic objective function values specified above. Qiskit currently does not support such experiments, however should this change in the future, there is potential for quantum distance metrics to be applied directly here, possibly leading to a greater embedding of the Safe Ml method into these systems.

After this is done the basic structure of the VQC can be imported form the Qiskit VQC library, which requires a feature map, a parameterized circuit, an optimization algorithm, a sampler, and a callback function.

Once this model has been instantiated, it is fitted on the training, and finally, the test data is passed to it for forming predictions.

For reproducibility, this is all compiled into one succinct function, seen below.

Algorithm 16 Make and Train a Variational Quantum Classifier (VQC)

```
Require: A set of features features and labels labels
Ensure: The trained VQC model and its performance metrics
  Set the random seed for reproducibility:
                    np.random.seed(123)
  Split the data into training and testing sets (80% train, 20% test):
  train_features, test_features, train_targets, test_targets ← train_test_split(features, labels, train_size=0.8,
  Define the ZZFeatureMap using the number of features $n$:
          feat_map ← ZZFeatureMap(feature_dimension=n, reps=1)
  Define the RealAmplitudes ansatz with $n$ qubits and 3 repetitions:
          ansatz ← RealAmplitudes(num_qubits=n, reps=3)
  Initialize the optimizer as COBYLA with a maximum of 50 iterations:
                optimizer ← COBYLA(maxiter=50)
  Initialize the sampler:
                    sampler ← Sampler()
  Initialize an empty list for tracking the objective function values:
                 objective_func_vals ← list()
  Create the VQC classifier using the defined feature map, ansatz, optimizer, and sampler:
  vqc ← VQC(feature_map=feat_map, ansatz=ansatz, optimizer=optimizer, sampler=sampler, callback=c
  Reset the objective function values list:
                 objective_func_vals ← list()
  Train the VQC using the training data:
               vqc.fit(train_features, train_targets)
  Calculate the VQC's accuracy on the training and testing data:
          vqc_train_score ← vqc.score(train_features, train_targets)
           vqc_test_score ← vqc.score(test_features, test_targets)
  Print the training and testing accuracy:
             Print("Training accuracy: ", vqc_train_score)
              Print("Testing accuracy: ", vqc_test_score)
  Get predictions from the trained VQC on the test data:
```

Quantum neural network

Given the highly variable structure of classical neural networks, let alone their quantum counterparts, these prove more complex to reusably implement than the variational quantum classifier. Such implementation rely more heavily on existing Qiskit infrastructure, which makes them more susceptible to future migrations rendering their layout obsolete.

To begin, these necessary libraries must be imported, and then, similar to the VQC, a feature map is produced. The dimensions of this feature map must align with the dimensions of the passed dataset in order to correctly convert the set into a quantum state.

A quantum circuit will then be instantiated, again ensuring this aligns correctly with he pre-specified dimensions of the output from the feature map.

In this project our focus is the common samplerQNN form of quantum neural network, and this can be produced with qiskits samplerQNN library. This requires a circuit, feature map, sampler, and parameter weights.

An optimizer is then chosen, and this is passed to a classifier alongside the QNN. as noted previously, future qiskit versions may possibly allow for custom optimization functions to be used here, which could potentially include ones which seek specifically to minimize the result of a given distance metric, allowing for more seamless integration of the quantum safe Ml method.

This classifier is then fit to the training data, and predictions are made on the test data.

Algorithm 17 Train and Evaluate a Quantum Neural Network (QNN) for Binary Classification

```
Require: A feature set $X$ and labels $y$ for binary classification
Ensure: The trained QNN classifier and its performance metrics
  Filter data to include only class 0 and class 1:
      $X \leftarrow X[y<2], \quad y \leftarrow y[y<2]$
  Standardize the features using StandardScaler:
      $\text{scaler} \leftarrow \text{StandardScaler()}$
      $X \leftarrow \text{scaler.fit\_transform}(X)$
  Apply PCA to reduce dimensionality to 2 components:
      $\text{pca} \leftarrow \text{PCA}(n\_components = 2)$
      $X \leftarrow \text{pca.fit\_transform}(X)$
  Encode binary labels (0 and 1):
      $y \leftarrow \text{np.array}([0 \text{ if label } == 0 \text{ else } 1 \text{ for label in } y])$
  Split the dataset into training and test sets (80% train, 20% test):
  $X\_train, X\_test, y\_train, y\_test \leftarrow \text{train\_test\_split}(X, y, test\_size = 0.2, random\_state = 42)$
  Define the Pauli feature map:
      feature_map ← PauliFeatureMap(feature_dimension=2, reps=3, paulis=['X', 'Z'])
  Define the variational ansatz:
      var_circuit ← RealAmplitudes(num_qubits=2, reps=3, entanglement='full')
  Initialize the sampler:
      sampler ← Sampler()
  Create the Quantum Neural Network (QNN):
  $\text{qnn} \leftarrow \text{SamplerQNN}(circuit = \text{feature\_map.compose(var\_circuit)}, input\_params = \text{feature\_map.parame}$
  Define the SPSA optimizer:
      optimizer ← SPSA(maxiter=300)
  Train the classifier using the training data:
      $\text{classifier} \leftarrow \text{NeuralNetworkClassifier}(qnn, optimizer = \text{optimizer})$
      $\text{classifier.fit}(X\_train, y\_train)$
  Predict on the test set:
      $y\_pred \leftarrow \text{classifier.predict}(X\_test)$
```

7.4 Incorporating SafeML Method

The proposed Quantum SafeMl method specifies applying the distance metrics to compare two different sets. the first of these is a set of predictions, wrongly classified into a given label, and the second of these is a set of predictions correctly classified into a given label. The value of the metric given from this calculation can then be compared to true accuracy of the model, in these cases possibly through use of a specified optimization function.

This can be done by instantiating the models and fitting them to a dataset as normal, but in the validation phase, recording the results of classification, and storing these int he appropriate data format, which in this case could either be a density matrix or a numpy array. These two sets will be computed using the appropriate metrics and compared to accuracy for a final verdict on classifier safety.

Below is flowchart showing the proposed quantum safeML method:

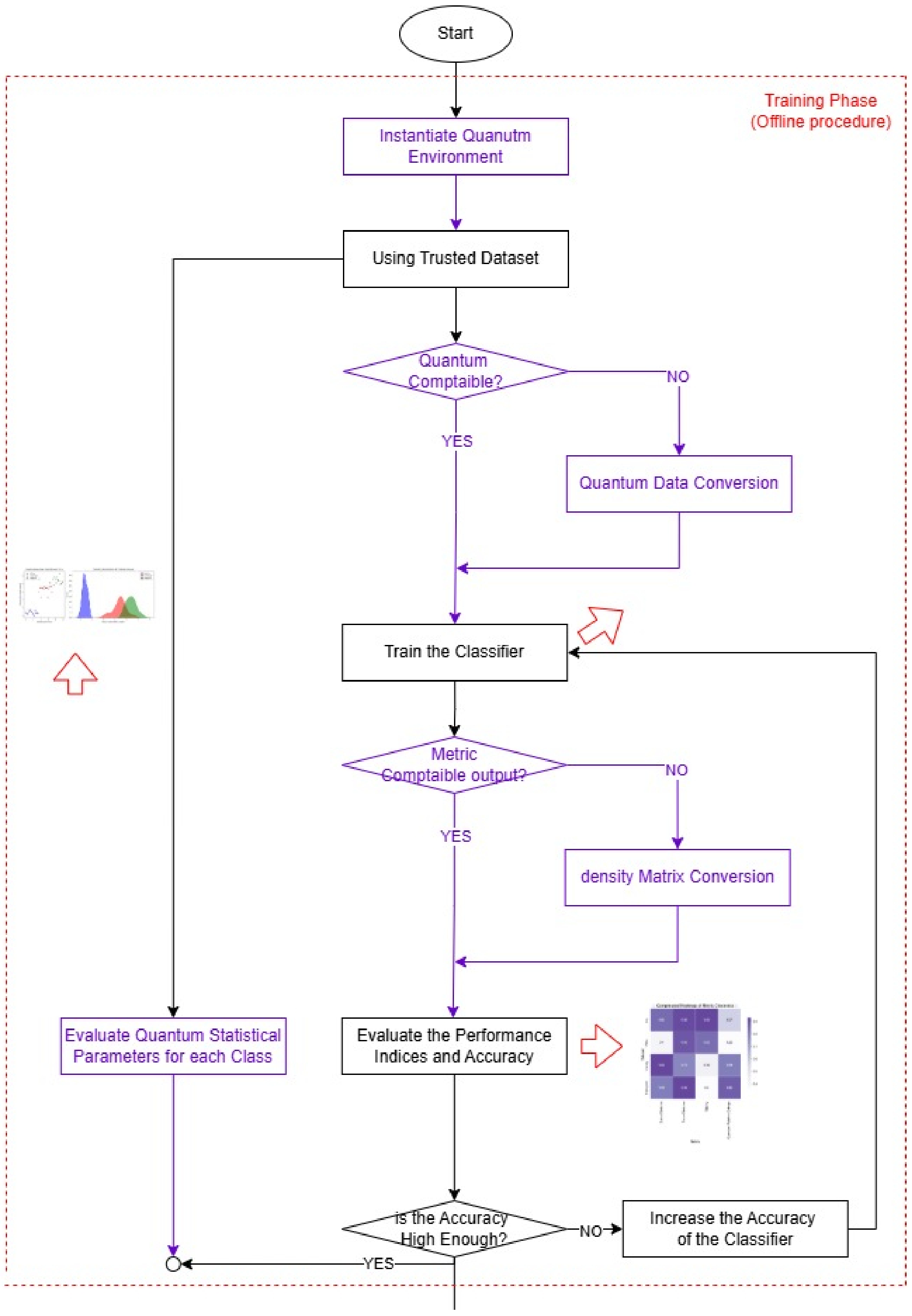

Start
Training Phase
(Offline procedure)
Instantiate Quanutm Environment
Using Trusted Dataset
Quantum Comptaible?
NO
YES
Quantum Data Conversion
Train the Classifier
Metric Comptaible output?
NO
YES
density Matrix Conversion
Evaluate Quantum Statistical Parameters for each Class
Evaluate the Performance Indices and Accuracy
is the Accuracy High Enough?
NO
Increase the Accuracy of the Classifier
YES

Figure 7.1 Proposed Q Safe Ml method(training phase)

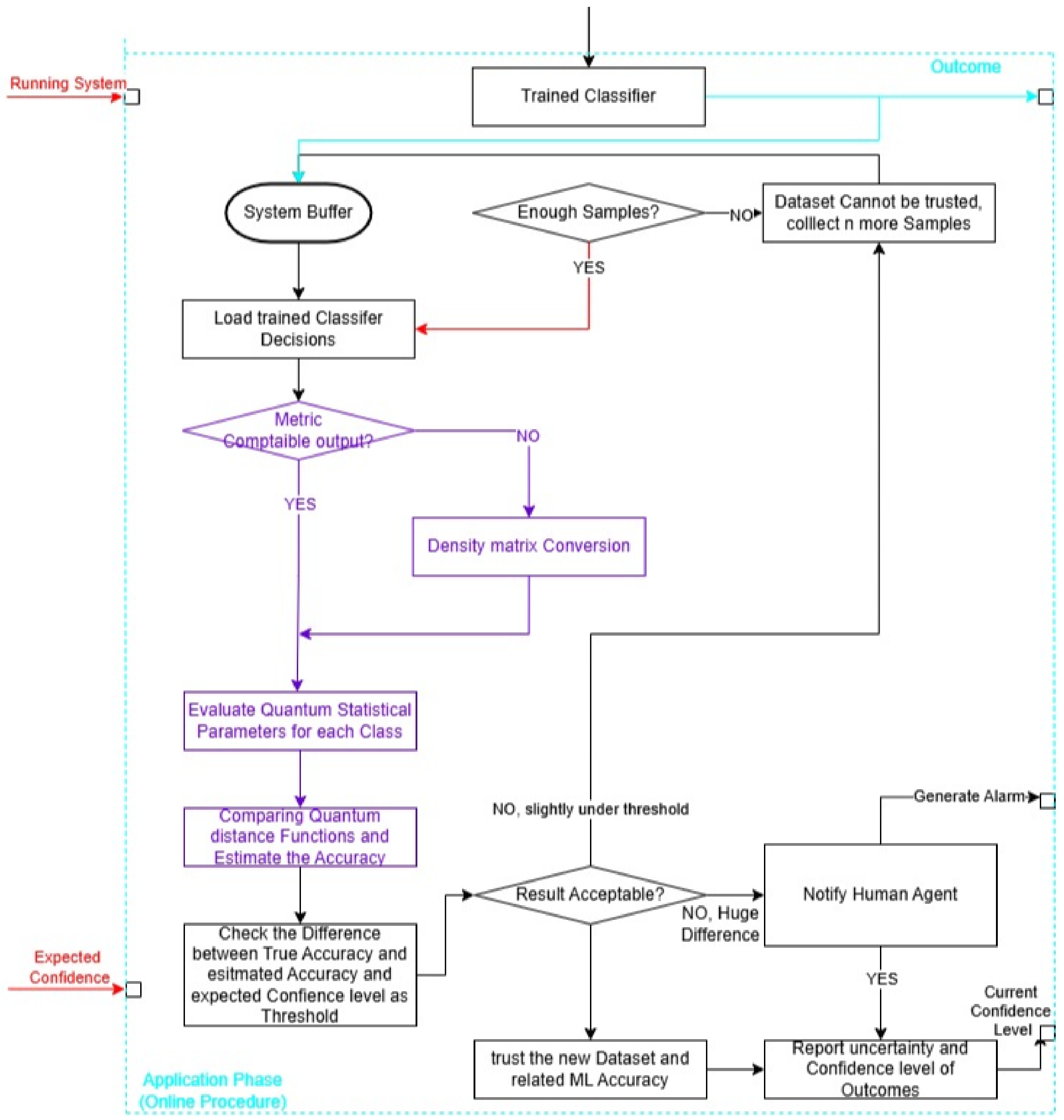

Running System
Trained Classifier
Outcome
System Buffer
Enough Samples?
NO
Dataset Cannot be trusted, colllect n more Samples
YES
Load trained Classifer Decisions
Metric Comptaible output?
NO
YES
Density matrix Conversion
Evaluate Quantum Statistical Parameters for each Class
Comparing Quantum distance Functions and Estimate the Accuracy
NO, slightly under threshold
Generate Alarm
Result Acceptable?
Notify Human Agent
NO, Huge Difference
Expected Confidence
Check the Difference between True Accuracy and esitmated Accuracy and expected Confience level as Threshold
YES
Current Confidence Level
trust the new Dataset and related ML Accuracy
Report uncertainty and Confidence level of Outcomes
Application Phase (Online Procedure)

Figure 7.2 Proposed Q Safe Ml method(online phase)

Chapter 8

Testing and Evaluation

Testing of the method can be done in different ways depending on the problem at hand. Two specific evaluations will be done. First, an example of the Q-SafeML method will be exercised on the designed implementation of a VQC inspired by the iris dataset notebook sourced from the Qiskit community pages.

A second example will be traversed through using a Quantum Convolutional Neural Network. This will follow the architecture discovered in experimentation from the Qiskit Community pages. This is done due to the quantum convolutional neural network being a close equivalent to the classical convolutional neural network used for the German traffic sign example of the traditional SafeML method.

8.1 Quantum SafeML on a VQC with Toy Dataset

In this section, the previously explored notebook which constructs a VQC model for operation on the iris dataset will be expanded to include classification on another similar dataset, the sklearn wine dataset. This set is slightly more complex, comprised of 13 features, and so it is expected that this will cause more errors for the classifier.

Additionally, for further comparison, other datasets will be included. After these models have been fitted and tested as normal, the output predictions will be used for the Quantum SafeML method. This is done through the use of custom dataset generation, where bunch objects of similar format to the sklearn datasets can be generated with random data points, but custom feature and label names. For transparency, the algorithm for generating these is included:

Algorithm 18 Generate Custom Synthetic Dataset

```
Require: class_labels: List of category names
Require: feature_names: List of numerical feature names
Require: num_samples: Number of samples to generate (default = 200)
Ensure: A dataset similar to sklearn's load_iris()
 1: Set random seed to 42 for reproducibility
 2: Initialize empty list features
 3: for each feature in feature_names do
 4:     Generate num_samples random values uniformly in range $[0,10]$
 5:     Append generated values to features
 6: end for
 7: Stack feature columns into feature_matrix
 8: Assign random class labels from class_labels to num_samples instances
 9: Encode labels numerically using LabelEncoder
10: Normalize feature values using MinMaxScaler
11: Return dataset as a structured object with:
12:         - Normalized data matrix
13:         - Encoded target labels
14:         - Class label names
15:         - Feature names
```

It should be noted that as this function generates datasets randomly, there will be no real expected connection between features and classes, and so the model is not likely to be effective at training and testing on these. The performance of the models is not of primary concern, as long as the Quantum SafeML method is able to be adequately deployed upon them after completion of training.

8.1.1 Training Model as Normal on Datasets

After applying appropriate preprocessing, the VQC is instantiated, using a feature map with a depth matching the number of features present in the dataset. Other features such as the sampler, as well as the callback and optimization functions, are likewise specified in this; however, for the purpose of this example, only the default values for these attributes are kept.

8.1.2 Predicting Model as Normal on Datasets

Once the model is fitted to the dataset, predictions are then run. Checking these predictions against the true set of labels is then done, and from this, the comparison sets will be defined.

8.1.3 Defining Sets for Comparison

For the Quantum SafeML method to operate, two separate datasets are required for comparison. These will be collected by selecting a specific class in the dataset (here, a generic class 1 is chosen for repeatability across all the different datasets, though this could be modified to be defined by specific class names), and compiling a set of predictions where this class was determined but was not correct, and another set where this class was correctly predicted. These two sets will be shaped and formatted accordingly and have the distance metrics applied on them, and these will be compared to the accuracy score of the model.

8.1.4 Instantiating Distance Metric Functions

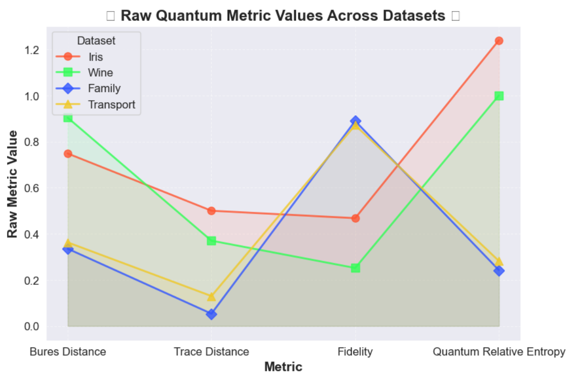
The distance metric functions of Bures distance, trace distance, fidelity, and quantum relative entropy are instantiated with respect to their designs and initial drafts in the experiments and development stage. Many of the data formatting safety measures employed in development will be utilized here, but for the purposes of visualizing our example, these may be edited to not give a code error, but rather assign an intentionally inflated value, such as 1, to data points of poor format, in order to keep visualizations while clearly signaling instances of data conversion requiring further formatting.

8.1.5 Results

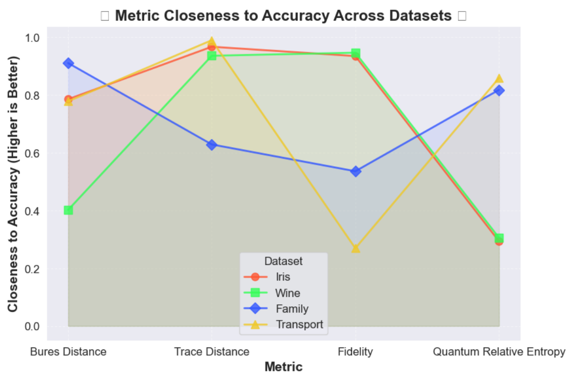
In order to derive results, each dataset's respective VQC generates a set of predictions on the test data. Following this, a particular class is chosen; in this example, it is always class 1. The predictions set is then traversed for instances of predictions into this class. Incorrect predictions of this class are put into the misclassified set, and correct predictions of this class are put into the correctly classified set. These are then formatted for distance metric calculation, and each of the four metrics is computed between these sets.

Dataset	Bures Distance	Trace Distance	Fidelity	Quantum Relative Entropy	True Accuracy
Iris	0.7482	0.5000	0.4677	1.2396	0.5333
Wine	0.9036	0.3701	0.2517	1.0000	0.3056
Family	0.3352	0.0535	0.8893	0.2424	0.4250
Transport	0.3627	0.1299	0.8714	0.2823	0.1406

Table 8.1 Quantum Metric Comparisons Across Datasets



(a) Raw Quantum Metric Values Across Dataset



(b) Metric Closeness to Accuracy Across Datasets

Figure 8.1 Comparison of Quantum Metric Values Across Datasets

In Table 8.1, we see how the metrics vary across the different datasets. Key points are quantum relative entropy values for the Iris and Wine datasets, as these exceed 1. As per the safety measures employed, this could imply matrices of invalid shape. In this case, this was caused by the classifier having no correct predictions for the specific classes of these datasets, which is expected given the basic running capacities given in the default model. A further development of the model architecture and insurance of positive definiteness mitigates this issue, but for now they are normalized results for following comparisons.

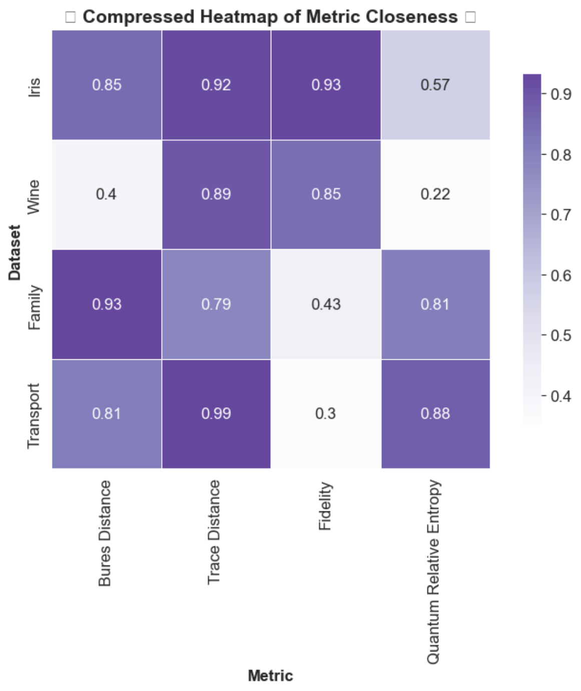


Figure 8.2 Heatmap of Metric Values Across Different Datasets on VQC

In Figure 8.1, it is observed that the metrics vary quite diversely across the different datasets, and further still they vary regarding their vicinity to accuracy of the model. This means the method is robust to exploring different facets of the data and computing a range of metrics upon it provided correct formatting.

In Figure 8.2, it is seen more clearly the variation of metric vicinity to model accuracy.

Overall, the VQC implementation functioned properly on the variety of datasets passed to it, to varying degrees of performance, and the metrics were able to determine a range of different results based on these performances.

8.2 Quantum SafeML on a QCNN

The German traffic sign dataset was explored but proved too complex for QCNN training, constrained by the capabilities of the quantum simulation. Instead, the application of Quantum SafeML onto QCNN is observed through yet another toy dataset.

The digits dataset from sklearn is comprised of 8x8 grayscale images that can occupy one of 10 classes representative of the digits 0 through 9 respectively. This is a far simpler dataset, and so it can be expected that our QCNN model will be more capable of making predictions here.

In the example QCNN, predictions are made on a dataset of synthetic images that are comprised of only 8 pixels, whereas this dataset is comprised of 64, meaning that whilst this is still simpler than the German traffic sign set, a reduction of dimensionality must be made in order to further ensure the model is capable of performing here.

To begin, it is ensured that the features are flattened from an image of 8x8 into a single-dimensional vector of 64 features. Then, further dimensionality reduction will be performed.

This is done through Principal Component Analysis. This allows us to reduce the dimensionality of our set of features to fundamental components, simplifying the data that the model will use to inform predictions.

In this case, PCA is used to reduce components from 64 to 6, which is then mapped to a circuit comprised of 6 qubits. Six is chosen as this can represent 64 features robustly.

Next, one-hot encoding is performed on the labels of the data, in order to ensure classification is done distinctly into classes.

Once the data has been adequately processed and formatted, the model architecture must be changed in key ways to ensure compatibility. This includes modifying the number of qubits in the circuit to be compatible with the number of features, and observables are ensured to match the 10 classes present in this dataset. The primary challenge of this conversion in model architecture was ensuring the model could transfer from a binary classification problem, seen in the example notebook for vertical vs. horizontal line classification, to a multi-class classification like the 10 digits in the sklearn dataset. Completion of this translation opens the doors for a simplistic and reusable function for generating QCNN models for multi-class classification problems.

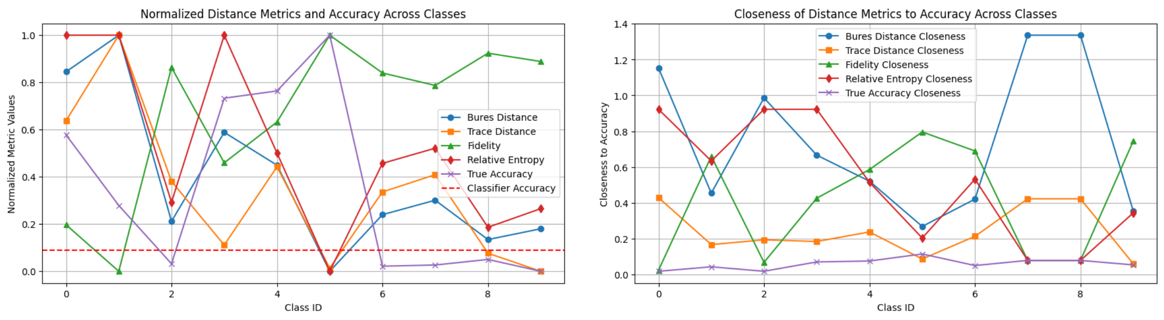
8.2.1 Quantum SafeML Method

The metrics are instantiated as before. A target class is again chosen. This class is used to create two sets of correctly labeled and incorrectly labeled data points.

These are then again converted into density matrices, compatible formats for the distance metric calculations, and these are compared to accuracy.

8.2.2 Results

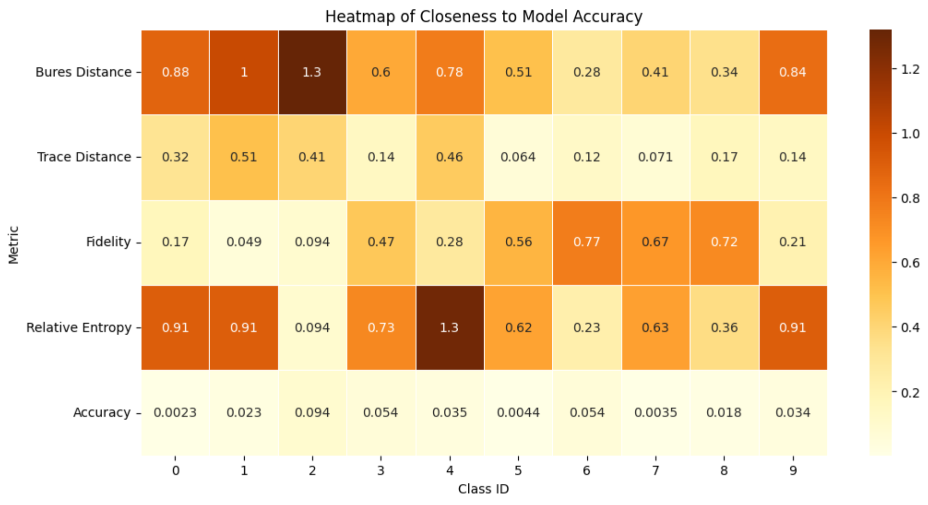
A different approach is taken to visualising method results in the QNN example, due to only one dataset being considered here. Instead, the wide range of different classes is utilised in order to see how the values for the metrics vary as different classes are targeted to create our sets of correct and wrong predictions.



(a) Metric Values for QNN Classes (b) Metric Accuracy Closeness in QNN

Figure 8.3 Comparison of QNN Metric Values and Accuracy Closeness

These graphs show a high variance of scores for each metric in each class case, and the accuracy vicinity further reflects this variation. It is interesting to note such a high distance from accuracy for Bures distance in classes 8 and 7.



Metric	0	1	2	3	4	5	6	7	8	9
Bures Distance	0.88	1	1.3	0.6	0.78	0.51	0.28	0.41	0.34	0.84
Trace Distance	0.32	0.51	0.41	0.14	0.46	0.064	0.12	0.071	0.17	0.14
Fidelity	0.17	0.049	0.094	0.47	0.28	0.56	0.77	0.67	0.72	0.21
Relative Entropy	0.91	0.91	0.094	0.73	1.3	0.62	0.23	0.63	0.36	0.91
Accuracy	0.0023	0.023	0.094	0.054	0.035	0.0044	0.054	0.0035	0.018	0.034

Figure 8.4 QNN Heatmap

This heatmap shows vicinity to accuracy across the classes and metrics, and it can be observed that some instances showcase a correlation with accuracy.

The constraints of the real hardware the simulations were run on limited the performance of the QNN model, but enhanced model architecture with more robust calculations—potentially from real quantum systems—could give way to further alignment between model accuracy and metric results from classes.

Overall, this again proved a diverse method of showing weaknesses in classifier prediction sets.

8.3 Employing SafeML on Quantum Kernel classifier

An additional test is done on the Quantum SafeMl method by applying it to a classier which uses Kernel Methods on ad hoc data.

8.3.1 Creating Ad hoc data

To begin, an ad hoc dataset is generated.

This algorithm allows for custom dimensionality of the dataset. in the above figure, it is observed how the different classes of the set occupy the feature space.

Algorithm 19 Generate Synthetic Dataset Using Qiskit's `ad_hoc_data`

```
Require: Training size $T = 50$, test size $S = 15$, number of features $n = 2$, margin gap = 0.1
Ensure: Training and testing features/labels and full dataset
    Set the number of features:
        $\text{adhoc\_dimension} \leftarrow 2$
    Generate the dataset using ad_hoc_data:
    $X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}}, D_{\text{total}} \leftarrow \text{ad\_hoc\_data}(T, S, n = \text{adhoc\_dimension}, \text{gap} = 0.1, \text{plot\_data} = \text{True}, \text{one\_ho}$
```

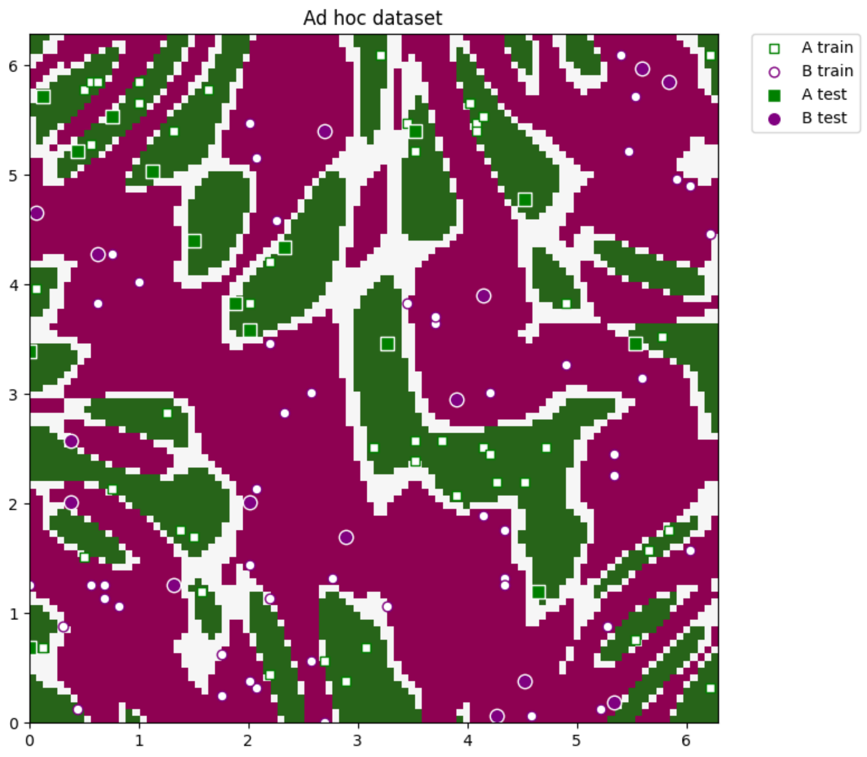


Figure 8.5 distribtuion of ad hoc data

8.3.2 Defining Quantum kernel

This entails encoding our ad hoc data into a quantum state through use of a feature map, here the ZZfeaturemap, and computes the similarity between the class states. Interestingly, it does this this through the use of fidelity, a quantum distance metric which is used in the Quantum SafeML method. This implies the data's compatibility with the range of quantum distance metrics the method employs at later stages.

Algorithm 20 Define Fidelity Quantum Kernel Using Qiskit Components

```
Require: Feature dimension $n = \text{adhoc\_dimension}$
Ensure: A fidelity-based quantum kernel object
  Define a ZZFeatureMap with linear entanglement:
  $\text{feature\_map} \leftarrow \text{ZZFeatureMap}(feature\_dimension = n, \text{reps} = 2, \text{entanglement} = \text{"linear"})$
  Initialize the quantum sampler:
  $\text{sampler} \leftarrow \text{StatevectorSampler()}$
  Define the fidelity estimator using Compute-Uncompute:
  $\text{fidelity} \leftarrow \text{ComputeUncompute}(sampler = \text{sampler})$
  Create the Fidelity Quantum Kernel:
  $\text{quantum\_kernel} \leftarrow \text{FidelityQuantumKernel}(fidelity = \text{fidelity}, feature\_map = \text{feature\_map})$
```

8.3.3 state vectors

Next, we use the State vector feature from qiskit quantum info, the constructing a function which will return the specific vectors of each of our quantum states through use of a new feature map.

8.3.4 training QSVC

QSVC stands for Quantum Support vector Classifier, and is a quantum extension of a regular Support vector Classifier. This type of QML model can be used for classification through the use of a quantum kernel, given here previously through the computation of fidelity. We instantiate the QSVC with this kernel and fit it tot he training data, then scoring it using the test data

Algorithm 21 Compute Statevectors for a Dataset Using a Feature Map

```
Require: Feature map circuit $\mathscr{F}$, dataset $X$
Ensure: List of statevectors corresponding to each data point in $X$
  Initialize empty list: statevecs ← []
  for each data point $x \in X$ do
      Assign parameters to feature map: circuit ← $\mathscr{F}.assign\_parameters(x)$
      Simulate the statevector: sv ← Statevector(circuit)
      Append sv.data to statevecs
  end for
  return statevecs
```

Algorithm 22 Train and Evaluate QSVC Using Fidelity Quantum Kernel

```
Require: Training features $X_{\text{train}}$, training labels $y_{\text{train}}$, test features $X_{\text{test}}$, test labels $y_{\text{test}}$,
  quantum kernel $\mathscr{K}$
Ensure: QSVC model and test classification accuracy
  Initialize the QSVC model with the quantum kernel:
          $$\text{qsvc} \leftarrow \text{QSVC}(quantum\_kernel = \mathscr{K})$$
  Train the QSVC model:
          $$\text{qsvc.fit}(X_{\text{train}}, y_{\text{train}})$$
  Evaluate accuracy on test data:
          $$\text{score} \leftarrow \text{qsvc.score}(X_{\text{test}}, y_{\text{test}})$$
  Print test score:
          Print("QSVC classification test score: ", score)
```

This fitting yields A classifications core of 0.93333, likely due to the small sample size of the dataset.

8.3.5 Threshold-Based Safety Monitoring in Quantum SafeML

In **Quantum SafeML**, we use various distance- or similarity-based metrics as indicators of model confidence. These metrics help determine whether a human agent should be alerted due to potential uncertainty or unsafe predictions made by the model. To facilitate this, we first define a **threshold value** that each metric will be compared against. If the metric exceeds this threshold, it may suggest that the model is operating in a region of low confidence, and a human-in-the-loop (HITL) intervention can be triggered.

Metric Normalization and Preprocessing

To ensure consistency across different metrics, we make several key adjustments:

- **Bures distance**, which ranges from 0 to $\sqrt{2} \approx 1.414$, is **normalized to the range [0, 1]** by dividing each value by $\sqrt{2}$. This aligns its scale with other normalized metrics.
- **Fidelity**, which naturally ranges from 0 (completely different states) to 1 (identical states), is **inverted** so that higher values indicate greater discrepancy, using the transformation $1 - \text{fidelity}$.

This preprocessing ensures consistency across all metrics, where a **higher value always indicates lower confidence or higher deviation**.

Threshold-Based Traversal and Human-In-The-Loop Alerts

Once metrics are normalized and aligned, we implement a thresholding mechanism. We traverse the dataset, evaluating the chosen metric(s) for each data point. If a metric exceeds the pre-defined threshold, the corresponding data point is flagged as **potentially unsafe** and appended to a list of unsafe samples.
This list can later be presented to a human agent for review, especially in high-stakes applications where model reliability is critical. For instance, in medical diagnostics or autonomous systems, it is essential to escalate low-confidence predictions to human experts for verification or override.

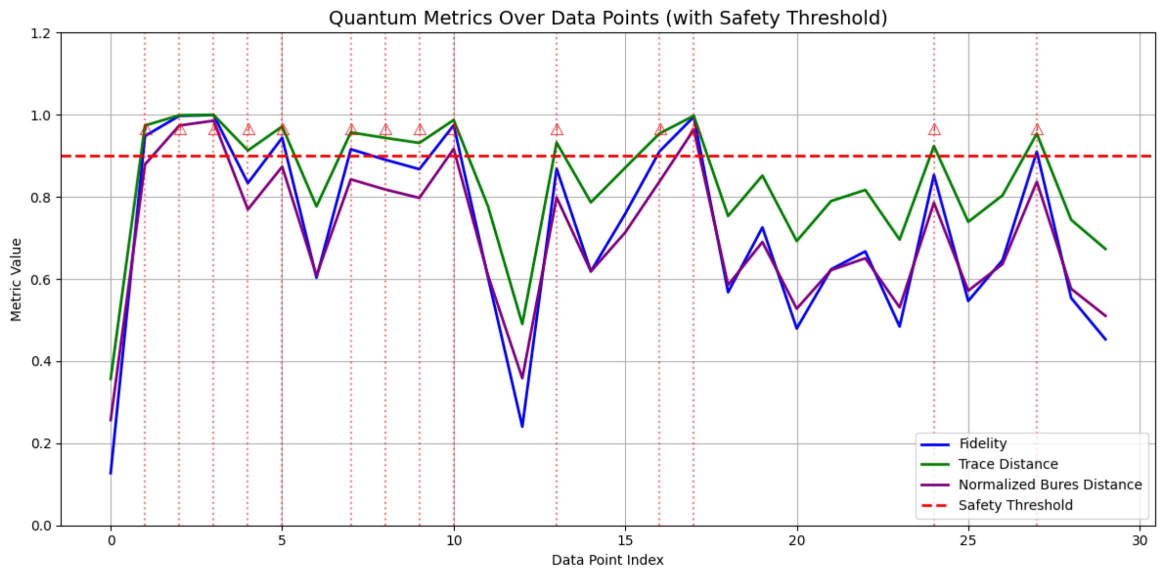


Figure 8.6 kernel method metric results compared to threshold value

Uncertainty and Trust in Quantum Models

High values in these discrepancy metrics (e.g., normalized Bures distance or inverted fidelity) often reflect that the model is:

- Encountering data that is **out-of-distribution**, or
- Operating near the **boundary of its decision function** where confidence is lower.

By identifying such instances through thresholding, we obtain a quantitative signal of **classifier uncertainty**. This enables dynamic monitoring of prediction safety and supports **human-machine collaboration**, where the model defers to human oversight when needed. **In summary**, this threshold-based approach transforms abstract quantum similarity metrics into actionable safety signals. By incorporating human-in-the-loop decision-making, we enhance both **trust** and **robustness** in quantum-safe machine learning pipelines.

Chapter 9

Results and Discussion

In this chapter, the proposed implementation of Quantum SafeML is expanded upon, and a discussion is had on the approach to the method, and how the project showcased its effectiveness.

9.1 Analysis of Findings: Where SafeML Works and Where It Doesn't

SafeML proved effective at providing a range of different performances across multiple distance metrics, allowing for ample comparison to the accuracy of a given classifier.

This proved particularly effective in VQC classification across the range of datasets and was tolerant to certain invalidates in some instances, such as quantum relative entropy in the sklearn toy datasets.

Applying the method onto a QCNN proved more challenging. This was due to the higher complexity of QCNN instantiation. The SafeML method remained effective at determining weaknesses of classifier predictions.

9.2 Metric Analysis

The specific metrics are evaluated, considering their ideal use cases, and how these applied to the navigated examples.

9.2.1 Trace Distance

Initially, we viewed this metric's ideal use case as comparing two states in a classifier, as its primary structure is best for distinguishability between two quantum states.

Application in the VQC

In the VQC, trace distance was able to find distinguishability between the states fairly consistently. Its highest raw value was recorded in the iris dataset, and this could be due to the low complexity of the iris dataset compared to the wine dataset, allowing for smaller overlap between predictions.

In terms of closeness to accuracy, trace distance saw the highest performance in the transport dataset, but was the consistently closest to the accuracy score across all metrics.

In 6.1, the closeness of accuracy for trace distance could be attributed to its clear and distinct fitting to the problem at hand. Trace distance is ultimately a measure of distinguishability between two quantum states, and so when we encode the set of incorrect predictions and correct predictions respectively for a given class into quantum states, trace distance can directly tell us how confused the classifier itself will be between its correct and incorrect predictions, from which it can be inferred how confident overall to be in the verdicts given by the model.

Application in the QNN

When assessed in the QNN, trace distance does not follow accuracy as well as when it was applied within the VQC results.

In 8.3b, it is observed that trace distance consistently lags behind Bures distance and quantum relative entropy across most of the assessed classes. However, it is notable that trace distance is more consistent within itself compared to these different metrics, and this could be due to the normalization measures applied on metric results in order to more directly compare them all to accuracy simultaneously.

Mathematical evaluation of Trace distance implementation

With trace distance between two density matrices ρ and σ, defined as:

$$D(\rho,\sigma) = \frac{1}{2}\|\rho - \sigma\|_1 = \frac{1}{2}\sum_i |\lambda_i|,$$

where $\|\cdot\|_1$ denotes the trace norm, and λ_i are the eigenvalues of the Hermitian matrix $\rho - \sigma$. the metric is intended to capture the total variation betwen the two quantum states.

When applied in the Quanutm SafeML method, in circumstances where QML classifier outputs were encoded from classical probability distributions of prediction labels into quantum states, trace distance $D(\rho,\sigma)$ provided a principled measure of how accurately the classifier's output reflected the target.

In our specific implementations, we defined matrices ρ and σ as the incorrect and correct predictions respectively for a given class in the data. Applying trace distance in this context, it is effectively measuring the extent to which incorrect predictions can be differentiated form correct ones, which could ease Interpretability in safety monitoring at runtime in a realised QML implementation.

An important feature of the trace distance is that it possesses an operational interpretation: it quantifies the maximum probability with which one can distinguish between ρ and σ using a single-shot measurement. Specifically, the success probability of optimal discrimination is given by:

$$P_{\text{success}} = \frac{1}{2} + \frac{1}{2}D(\rho,\sigma).$$

This makes the trace distance particularly valuable for QML evaluation, as it ties directly to the classifier's distinguishability performance in terms of quantum measurements.

In experimental or simulation settings, a small trace distance (e.g., $D(\rho,\sigma) \approx 0$) indicates high fidelity between the predicted and true quantum states, suggesting effective learning and generalization by the QML model. Conversely, larger values approaching 1 highlight misclassifications or poor generalization.

Thus, the trace distance acts as a powerful tool for validating and benchmarking quantum classifiers, especially in tasks where outputs are naturally encoded in quantum states.

Application of Trace Distance in Real Applications

The examples in this project showcased trace distance's capability in proof-of-concept application of QML models. However, in real-world applications, it would likely fare best in similar such applications, provided adequate data preprocessing and normalization, as well as classifier training, is completed.

Trace distance, being a metric of how distinguishable two quantum states are, lends itself directly to the concept of Quantum SafeML, and so when applied to a quantum classifier, it has potential to be very effective at assessing the difference between sets of predictions based on their validity. This was seen in its good performance in the VQC notebook, although its limitations in the QNN example possibly speak to its susceptibility to severely overfitting classifiers.

9.2.2 Bures Distance

Initially, we viewed this metric's ideal use case as measuring robustness in noisy quantum classifiers, as its primary structure is best for very mixed quantum states. The ideal scenario for this metric within the context of the Quantum SafeML method is of fully realized real quantum systems, which at this time and likely for the foreseeable future will be highly susceptible to external noise factors.

Application in the VQC

In the VQC, Bures distance had a very high proximity to overall classifier accuracy across most of the explored datasets, with a notable outlier being the wine dataset, seen in 6.1. This could be due to the relatively high complexity of the wine dataset, possessing multiple classes and features which could have obscured the classifier's confidence compared to the more simple iris dataset.

Application in the QNN

When assessed in the QNN, Bures distance performs better than most metrics when considering proximity to accuracy.

In 8.3b, it is observed that Bures distance performs highly alongside quantum relative entropy across most of the assessed classes. However, it is notable that the results of Bures distance within itself show high inconsistency, and this could be due to Bures distance's susceptibility to noise.

The primary concern of quantum computing, and the reason it is not readily commercially available at this time, is its highly sensitive nature and heavy requirement of very specific conditions to run in, which can be obscured very easily by even the smallest external factors.

In the simulation used here, it is possible that Bures distance was highly sensitive to the specific ways in which the QCNN encoded the different classes, and so this led to inconsistent outputs.

Mathematical evaluation of Bures distance implementation

We defined The Bures distance between two density matrices ρ and σ as:

$$D_{\text{Bures}}(\rho,\sigma) = \sqrt{1 - F(\rho,\sigma)},$$

where $F(\rho,\sigma)$ is the **fidelity** between ρ and σ, defined as:

$$F(\rho,\sigma) = \left(\text{Tr}\left(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)\right)^2.$$

The fidelity $F(\rho,\sigma)$ is a measure of how similar the two quantum states are, with $F(\rho,\sigma) = 1$ indicating that the states are identical, and $F(\rho,\sigma) = 0$ indicating that the states are orthogonal.

In our QML case studies, with the quantum states ρ and σ representing predictions made by a classifier to a given target, bures distance $D_{\text{Bures}}(\rho,\sigma)$ essentially measured how well the classifier's incorrect predictions of the class approximated those same class predictions in instances o correctness. For safety monitoring, this provides an insight into the confusion that could be made between poor predictions and valid ones, which could further imply a measure in place for notifying human agents of poor distinguishability.

One of the advantages of the Bures distance over the trace distance is its continuous nature and its ability to capture more subtle differences between states. It is closely related to the quantum Fisher information and provides a smoother measure of distinguishability compared to the discrete nature of the trace distance. WHilst this does not lend itself as neatly to classification problems, it does provide another valuable insight into quantum state differentiation.

The Bures distance is especially useful when assessing quantum classifiers in scenarios where the quantum states are not strictly diagonal or where they arise from entangled systems. It captures correlations and coherences present in the quantum state, which the trace distance might miss.

In practice, a small Bures distance (i.e., close to 0) indicates that the quantum classifier has successfully learned the underlying distribution, while a large Bures distance (closer to 1) indicates poor classification performance. Thus, the Bures distance is a powerful metric for evaluating the performance of quantum classifiers, particularly in tasks where quantum coherence and entanglement play a critical role.

$$D_{\text{Bures}}(\rho,\sigma) = \sqrt{1 - F(\rho,\sigma)} = \sqrt{1 - \left(\text{Tr}\left(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)\right)^2}.$$

Application of Bures Distance in Real Applications

Due to Bures distance's focus on mixed states, it makes it uniquely prepared for handling cases of highly noisy classifications.

This will prove useful in the infantile stages of real quantum hardware development, while susceptibility to noise will still be a primary concern. Whilst Bures distance cannot directly mitigate the effect this noise will have on QML predictions, the variation of its results across different classes could be used to infer a model's overall susceptibility to noise.

9.2.3 Fidelity

Initially, we viewed this metric's ideal use case as evaluating qGAN performance, as its primary structure is best for overlap in quantum states.

Application in the VQC

In the VQC, fidelity performed relatively poorly compared to the other distance metrics applied.

In 6.1, it was seen that fidelity had particularly low closeness to the scores of the family and transport datasets.

Of course, these datasets were completely synthesized and were only implemented to verify the VQC model's adaptability to differently shaped datasets. However, the metrics applied to the classifier results in these instances should still be able to adequately compute on even these artificial sets.

The other metrics saw consistency between the real toy datasets and the synthesized ones, and so it cannot be inferred that the poor performance of the fidelity metric can be attributed to the origin of these specific sets.

Instead, it is likely that the classifier's encoding of these classes for these datasets was different.

Even with the same variational architecture, if a certain dataset has well-separated class features, the model can create tight quantum states and fidelity will work well to distinguish correct from incorrect.

However, if the data in another dataset is noisy, overlapping, or not separable in the model's encoding scheme, the resulting quantum states may not be cleanly distinguishable, so fidelity becomes noisy or uninformative.

Application in the QNN

In the QNN, fidelity saw moderate success.

In 8.3b, it was observed that it performed better than other metrics in classes 1, 4, 5, 6, and 9, in terms of closeness to model accuracy.

However, in classes 7 and 8 it severely underperformed in accuracy proximity. This is likely not due to an issue in the content within these classes itself, but rather, due to added safety measures instantiated in our fidelity implementation, which ensured to attribute a value of 1 in order to signal issues with the inputted sets.

The high fidelity values for classes 7 and 8 speak to respective instances where this measure was employed, but it is unclear why specifically these classes experienced an issue during the application of the metric after model predictions. It is likely just how this specific instance of the model predicted certain states in this run.

This draws attention to the risk of Quantum SafeML being restricted by the specific ways the models happen to train in given instances, and this could lead to further safety measures in place in QML implementations, such as training multiple classifiers in parallel on data, in order to ensure tolerance to one model's poor encoding of data.

Mathematical Evaluation of Fidelity

For two density matrices, ρ and σ, the fidelity is defined as:

$$F(\rho,\sigma) = \left(\mathrm{Tr}\left(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)\right)^2.$$

WHich measures overlap between the two matrices. It ranges from 0 to 1, with the following interpretation: - $F(\rho,\sigma) = 1$ indicates that the two quantum states are identical. - $F(\rho,\sigma) = 0$ indicates that the quantum states are orthogonal, meaning there is no overlap between them.

In our case studies, the fidelity metric was applied tot he states represneting encoded incorrect and correct classifications for a given label.

The fidelity measure was useful for the mixed quantum states, providing a robust measure of how well these mixed states approximate the true target.

Fidelity is closely related to distance metrics such as the Bures distance and trace distance. The Bures distance is directly derived from the fidelity:

$$D_{\text{Bures}}(\rho,\sigma) = \sqrt{1 - F(\rho,\sigma)}.$$

Bures distance can be viewed as a complement of the fidelity. When the fidelity is high (close to 1), the Bures distance is small, indicating that the quantum states are very similar. Conversely, when the fidelity is low (close to 0), the Bures distance is large, signaling that the states are significantly different.

One of the advantages of using fidelity in the Quantum SafeMl method was that it naturally accounts for quantum coherences and correlations present in the quantum states. This makesit valuable when entanglement and superposition play a key role in the representation of data.

Fidelity can also be seen as an indicator of how well a quantum model has learned a given task. In the ideal scenario, the predicted quantum state σ would perfectly match the true quantum state ρ, leading to a fidelity of 1. However, in practical scenarios with noise, imperfections, and other quantum effects, the fidelity provides a clear metric to assess the degree of approximation.

For example, if ρ is the true state (e.g., representing the true class label) and σ is the predicted state (e.g., representing the classifier's prediction), a high fidelity value suggests that the quantum classifier has correctly learned to map inputs to their corresponding class labels. Conversely, a low fidelity suggests that the classifier's output significantly deviates from the true target.

In summary, the fidelity metric is a highly effective tool for evaluating quantum classifiers, as it provides a clear and interpretable measure of how well the predicted quantum state matches the target state. It is especially useful in situations where the quantum states are probabilistic or mixed, as it captures the overall similarity between the states regardless of whether they are pure or mixed.

$$F(\rho,\sigma) = \left(\mathrm{Tr}\left(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)\right)^2.$$

Application of Fidelity in Real Applications

Fidelity was initially thought to be ideal for qGAN situations.

qGAN models are quantum equivalents to quantum adversarial networks and are another hybrid model. They are comprised of a quantum generator, which outputs a quantum state that when measured gives samples approximating the target distribution, and a classical discriminator, which tries to distinguish between real data (from the target distribution) and fake data from the quantum generator. These components are trained adversarially in order to ultimately encode a classical distribution into quantum circuits so that we can sample from it efficiently.

Such a model was not implemented as an example in this project, due to the high computational cost required for adversarial training loops.

It can, however, be seen that fidelity would be suited to such models. This is because in qGANs, you're training a quantum generator to reproduce a target distribution, assessing how similar the output quantum state is to the real one.

Fidelity is the natural quantum analog of comparing two probability distributions directly. Unlike other metrics, fidelity focuses on the similarity of states directly, which is often the real question in generative tasks, especially during the model adversarial training phase.

This makes fidelity very effective at determining how close two distributions really are, assuming that the training was not faulty and encoding of data by the generator was done appropriately. If there was an error in the encoding, the fidelity metric would likely output an extreme value (such as 0 or 1) due to how closely fidelity relates to raw overlap between states.

9.2.4 Quantum Relative Entropy

Initially, this metric was envisioned for cases of quantum data privacy, as its primary structure is best for asymmetric comparisons of quantum states.

Application in the VQC

In the VQC, quantum relative entropy produced the most variance across its different results. Its most notable deviation was in the iris dataset, where it recorded a value of 1.0.

This is an unusual result which, upon inspection, stemmed from the inputted states to the metric evaluation having no shared basis — meaning the metric couldn't be computed in a typical fashion and instead returned a value of 1 as a signal of divergence.

This shows the limitation of quantum relative entropy as a metric: it can only function correctly when both states share the same support. If not, it breaks down.

Across the other datasets, QRE behaved more reasonably, though with slightly more noise than other metrics, likely due to this fundamental limitation and its dependence on logarithmic operations.

Application in the QNN

In the QNN, QRE was among the highest-performing metrics in terms of accuracy proximity. As seen in 8.3b, QRE closely matched accuracy across most of the classes. This shows that in more complex classifiers such as the QCNN, where internal representations are harder to disentangle cleanly, the asymmetric nature of QRE might help in distinguishing directional divergence between correct and incorrect prediction states.

However, just like in the VQC, care must be taken in how the metric is applied. If the classifier ever encodes a prediction class into a state orthogonal to its counterpart class, QRE can explode in value or become undefined.

This is not ideal for safety analysis — and this risk should be accounted for when using QRE in real systems.

Mathematical evaluation of QRE

For two density matrices ρ and σ, the quantum relative entropy is defined as:

$$S(\rho||\sigma) = \mathrm{Tr}\left(\rho \log \rho - \rho \log \sigma\right).$$

The relative entropy is a non-symmetric measure, meaning $S(\rho||\sigma) \neq S(\sigma||\rho)$. This metric captures the asymmetry of the difference between the quantum states ρ and σ.

A high quantum relative entropy indicates that the classifier's incroeect predictions σ is quite different from the correct prediction state ρ. In contrast, a lower value suggests that the classifier's predictions are easily confused.

The quantum relative entropy is closely related to trace distance and fidelity. While it is not a true distance metric (due to its asymmetry), it provides valuable insights into the difference between two quantum states. Unlike the trace distance, which measures the total variation between two states, or fidelity, which measures overlap, this metric is a more general and non-symmetric measure of distinguishability.

In a quantum classification model contexts, the quantum relative entropy can be used to quantify the misclassification error or prediction uncertainty. For example: - A low relative entropy indicates that the quantum classifier has accurately predicted the class, meaning that the predicted state σ closely matches the true state ρ. - A high relative entropy signals a significant divergence between the predicted state and the true class, suggesting a poor performance by the classifier.

The quantum relative entropy serves as a powerful tool to measure the divergence between predicted quantum states and the true target states or between correct and incorrect classifier predictions. It is particularly useful in settings where asymmetry between states is important, and it provides a more general measure of distinguishability than other metrics like fidelity or Bures distance.

In quantum classification tasks, the quantum relative entropy allows us to assess how well the predicted quantum state represents the true class, and can help drive optimization algorithms towards minimizing the misclassification error.

$$S(\rho||\sigma) = \mathrm{Tr}\left(\rho \log \rho - \rho \log \sigma\right).$$

Application of QRE in Real Applications

In quantum data privacy, QRE has been proposed as a foundational metric. This is because QRE, like classical relative entropy, measures how much one probability distribution diverges from another — but extended to the quantum setting.

This has value in contexts such as differential privacy for quantum datasets, where measuring how much extra information is gained (or leaked) from one state compared to another is key.

For practical Quantum SafeML usage, QRE may be useful in monitoring classifier robustness in adversarial environments, particularly where there is a clear expected or "safe" quantum state that new predictions should not deviate from too far.

Its directional nature — i.e., $D(\rho||\sigma) \neq D(\sigma||\rho)$ — can be used to emphasize shifts in safety-critical states, and could prove invaluable in supervised systems with asymmetric tolerance for error (e.g., misclassifying a 'normal' case as 'faulty' is not as bad as the reverse).

9.3 Final Thoughts on Metric Selection

When choosing a metric for Quantum SafeML, a central consideration should be the specific goal of the safety validation process.

- **Trace Distance** offers the most consistent and interpretable indicator of classifier confusion and performs reliably in most VQC use cases.
- **Bures Distance** shines when applied to noisy or highly mixed states, particularly in realistic quantum hardware scenarios where decoherence is significant.
- **Fidelity** is ideal in generative tasks or overlap assessments, such as qGANs or other quantum simulators, but is vulnerable to encoding quirks and data entanglement.
- **Quantum Relative Entropy** is powerful for asymmetric error evaluation and privacy-related divergence, but must be applied carefully to avoid invalid inputs.

The ultimate takeaway is that no single metric is "best" universally. Each has a strength, a bias, and a failure mode. A well-rounded Quantum SafeML system would ideally implement multiple metrics in parallel and analyze their convergence or divergence on classifier predictions.

9.4 Kernel method Showcasing agent monitoring

In the kernel experiment, an exploration is made of further expanding the meaning derived from the chosen distance metrics, beginning with normalizing and inverting the metric values where necessary to have them all in a consistent space and interoperability.
This is then compared to a reference threshold, and this is used for highlighting the exact datapoints which cause concern in the safety monitoring stage.
This could be expanded in future works with dynamic thresholding, which could be used to, at runtime, define the exact parameters that categorized poor model confidence in certain systems.
This overall showcased how the method could be deployed with respect to notifying human agents, at least provisionally.
The exact way fo notifying human agents will vary from system to system, and the format that is appropriate given the scenario. however, the fact that the method is able to provide a wide range of different ways of identifying these potentially unsafe predictions means there is adequate infrastructure in place within this method for catching poor predictions.

9.5 Quantum Hardware Considerations

While this project used simulators for both the VQC and QNN examples, future work will require benchmarking on real quantum hardware.

This introduces new noise sources: gate errors, measurement errors, crosstalk, and environmental decoherence. Metrics like Bures Distance and QRE, which are inherently noise-sensitive, might see even greater importance in this setting.

Simulators assume ideal gates unless noise models are explicitly added, so results in this project may be optimistic relative to real-world deployments.

Furthermore, quantum circuit depth and connectivity constraints will limit SafeML implementation in NISQ devices. This makes it crucial to continue researching low-depth, resource-efficient encodings and evaluation metrics.

9.6 Limitations

Multiclass classifiers introduce new challenges for quantum metric evaluation, especially in state preparation (how to encode multiple class prediction groups into valid quantum states).

Also, the metrics were only applied post hoc — after predictions. A more tightly integrated approach might involve continuous monitoring of model states during training, or even using metric-based loss functions to encourage more robust encodings.

Lastly, due to simulator constraints, many practical realities of live quantum environments (latency, calibration, etc.) were not reflected in this study.

9.7 Potential expansion - quantum SafeML as a service

It could be conceived that this method could be built as a bespoke application for industrial systems should such systems be viable int he near future
this opens the door to the possibility fo Quantum safeMl as a service, thoguh the exact ways of doing this could take many forms.

First, it could be as a public python package. A local package was made for this project for testing the versatility of the implemented functions, though this method is built heavily from qiskit and so has many dependencies.

it could further be speculated that the deployment could included the classifier model built in, and automatically assesses safety as part of model architecture, however, this would be a huge expansion of the method, encapsulating much more of the system.

9.8 Conclusion

This chapter explored the behavior of different quantum distance metrics when applied within the Quantum SafeML framework.

It was found that the metrics had distinct strengths and weaknesses, and no single solution fit all classifier types or data sets. Trace distance was the most stable, Bures distance most suited for noisy settings, fidelity best in generative models, and quantum relative entropy optimal for asymmetric divergence.

As quantum computing continues evolving, safety and reliability will remain key concerns. Methods such as Quantum SafeML — supported by thoughtful metric selection — will become essential tools in ensuring QML models remain interpretable, trustworthy, and robust in deployment.

Bibliography

[1] Adepoju, O., et al. (2019). *Quantum Computing: A Paradigm Shift in Computational Technology*.

[2] IBM (2024). Qubit. [online] *IBM*. Available at: https://www.ibm.com/think/topics/qubit.

[3] Neven, H. (2024). Meet Willow, our state-of-the-art quantum chip. [online] *Google*. Available at: https://blog.google/technology/research/google-willow-quantum-chip/.

[4] IBM (2024). IBM roadmap to quantum-centric supercomputers (Updated 2024). *IBM Quantum Computing Blog*. Available at: https://www.ibm.com/quantum/blog/ibm-quantum-roadmap-2025?lnk=ushpv18r1.

[5] Ladd, T.D., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., and O'Brien, J.L. (2010). Quantum computers. *Nature*, **464**(7285), pp. 45–53. doi: https://doi.org/10.1038/nature08812.

[6] Ahmed, T., Kashif, M., Marchisio, A., & Shafique, M. (2025). Quantum Neural Networks: A Comparative Analysis and Noise Robustness Evaluation. Retrieved from https://arxiv.org/abs/2501.14412.

[7] Benedetti, M., Lloyd, E., Sack, S., & Fiorentini, M. (2019). Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, **4**(4), p.043001. doi: https://doi.org/10.1088/2058-9565/ab4eb5.

[8] Gil Fuster, E. M. (2018). Variational Quantum Classifier. Retrieved from https://diposit.ub.edu/dspace/bitstream/2445/140318/1/GIL%20FUSTER%20Elies%20Miquel.pdf.

[9] Schuld, M., Sweke, R., & Meyer, J.J. (2020). The effect of data encoding on the expressive power of variational quantum machine learning models. *Physical Review A*, **102**(3), p.032420. doi: https://doi.org/10.1103/PhysRevA.102.032420.

[10] Liang, S., Li, Y., & Srikant, R. (2020). Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv:1706.02690 [cs, stat]*. Available at: https://arxiv.org/abs/1706.02690.

[11] Zolfagharian, A., Abdellatif, M., Briand, L.C., & S, R. (2025). SMARLA: A Safety Monitoring Approach for Deep Reinforcement Learning Agents. *IEEE Transactions on Software Engineering*, **51**(1), pp.82–105. doi: https://doi.org/10.1109/tse.2024.3491496.

[12] Aslansefat, K., Sorokos, I., Whiting, D., Kolagari, R. T., & Papadopoulos, Y. (2020). SafeML: Safety Monitoring of Machine Learning Classifiers through Statistical Difference Measures. *arXiv*. Retrieved from https://arxiv.org/abs/2005.13166.

[13] Xu, Z., & Saleh, J.H. (2021). Machine learning for reliability engineering and safety applications: Review of current status and future opportunities. *Reliability Engineering & System Safety*, **211**, p.107530. doi: https://doi.org/10.1016/j.ress.2021.107530.

[14] Paleyes, A., Urma, R.-G., & Lawrence, N.D. (2022). Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Computing Surveys*, **55**(6). doi: https://doi.org/10.1145/3533378.

[15] Kim, B.C., Kim, B., & Hyun, Y. (2024). Investigation of out-of-distribution detection across various models and training methodologies. *Neural Networks*, **175**, p.106288. doi: https://doi.org/10.1016/j.neunet.2024.106288.

[16] European Union (2024). The EU Artificial Intelligence Act: Regulatory Framework for Trustworthy AI. Retrieved from https://artificialintelligenceact.eu.

[17] ieeexplore.ieee.org. (n.d.). Quantum Vs Classical Computing: a Comparative Analysis | IEEE Conference Publication | IEEE Xplore. Retreived from https://ieeexplore.ieee.org/document/10062753