Quantum Cybersecurity Analytics:
Evaluation of Hybrid QML for DGA Botnet Detection

by Madjid Golparvaran Tehrani

B.Sc. in Computer Engineering-Software, May 2010, Azad University
M.Sc. in Advanced Computer Science, May 2016, University of Leicester

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

May 19, 2024

Praxis directed by

Raymond Chow
Professorial Lecturer of Engineering Management and Systems Engineering

The School of Engineering and Applied Science of The George Washington University certifies that Madjid Golparvaran Tehrani has passed the Final Examination for the degree of Doctor of Engineering as of January 19, 2024. This is the final and approved form of the praxis.

Quantum Cybersecurity Analytics:
Evaluation of Hybrid QML for DGA Botnet Detection

Madjid Golparvaran Tehrani

Praxis Research Committee:

Dr. Amir Etemadi, Associate Professor of Engineering and Applied Science, Committee Chair

Dr. Raymond Chow, Professorial Lecturer of Engineering and Applied Science, Committee Member

Dr. J.P. Blackford, Professorial Lecturer of Engineering and Applied Science, Committee Member

© Copyright 2024 by Madjid Golparvaran Tehrani
All rights reserved

Dedication

In the loving memory of my late father, whose enduring spirit and tireless support for my educational pursuits live within these pages. Each achievement documented here mirrors his legacy of love and dedication. To my steadfast wife and daughter, pillars of love and support through tumultuous times, this work stands testament to the sacrifices made and challenges overcome together.

Acknowledgments

The author extends heartfelt gratitude to the advisory committee for their invaluable guidance and constructive feedback throughout the development of this praxis. Further appreciation is extended to the faculty of George Washington University for initiating the first cohort in cybersecurity analytics, thereby providing the broader academic context within which this research is situated. Heartfelt gratitude is extended to Prof. Dr. Shahram Sarkani for the invaluable guidance and unwavering support that illuminated the path to completing my education with wisdom and clarity. Acknowledgment is also due to my supervisor, Prof. Dr. Raymond Chow, a nurturing force for my creative and boundless intellectual exploration, providing a supportive environment for venturing confidently into quantum cybersecurity analytics.

A special note of appreciation is reserved for Dr. Chelsie Thompson, whose assistance was crucial in gaining access to Azure Quantum devices during a critical phase where access to quantum computers was a formidable challenge. Her support and Microsoft's Azure Quantum team paved the way for pivotal breakthroughs and progress in this endeavor.

Special appreciation for Prof. Dr. Bill Buchanan OBE and Dr. Eldar Sultanow is necessary. With Prof. Buchanan's encouragement and unparalleled expertise in computer science as a guiding force, and Dr. Sultanow's steadfast belief in my potential serving as motivation, their contributions have been indispensable.

Abstract of Praxis

Quantum Cybersecurity Analytics: Evaluation of Hybrid QML for DGA Botnet Detection

This praxis presents a comprehensive investigation into the feasibility and speed of quantum cybersecurity analytics during the noisy intermediate scale quantum (NISQ) era. The research utilizes simulators and quantum devices from IonQ, Rigetti, Quantinuum, and the AER simulator. Classical and quantum binary classifiers are employed to explore the potential of quantum machine learning for achieving quantum advantage in cyber defense. Specifically, the praxis focuses on the detection of domain generation algorithm (DGA) botnets, a technique commonly used in distributed denial-of-service attacks. hybrid quantum binary classifiers (HQBC) such as quantum neural networks, quantum support vector classifiers (QSVC), Pegasos: Primal estimated sub-gradient solver for SVC, and variational quantum classifiers (VQC) are tested on the DGA Botnet cybersecurity dataset. The research identifies and analyzes the respective limitations and bottlenecks of these classifiers. Furthermore, this research introduces a novel algorithm, the quantum Hoeffding tree classifier (QHTC). This algorithm applies the concept of online incremental learning to quantum algorithms through tree-based classification. Notably, QHTC outperforms existing algorithms in terms of both accuracy and speed, avoiding the optimization trap. The findings of this praxis suggest promising avenues for leveraging quantum computing capabilities in enhancing cybersecurity measures, even within the constraints of NISQ device's limitations.

Table of Contents

Dedication ..... **iv**

Acknowledgments ..... **v**

Abstract of Praxis ..... **vi**

List of Figures ..... **ix**

List of Tables ..... **x**

Chapter 1 - Introduction **1**

1.1 Background 1

1.2 Research Motivation 3

1.3 Problem Statement 5

1.4 Thesis Statement 5

1.5 Research Objectives 5

1.6 Research Questions and Hypotheses 5

1.7 Scope of Research 6

1.8 Research Limitations 11

1.9 Organization of Praxis 12

Chapter 2 - Literature Review **13**

2.1 Botnets 13

2.2 Quantum Machine Learning 17

2.3 Optimizers in Quantum Machine Learning 29

2.4 Domain Generation Algorithm Botnets, Quantum Machine Learning, and the Knowledge Gap 39

Chapter 3 - Methodology **41**

3.1 Introduction 41

3.2 Prerequisite Dataset 43

3.3 Classic Binary Classifiers as Benchmarks 46

3.4 Simulators, Real Devices, and the Role of Optimizers 47

3.5 Selection of Hybrid Quantum Binary Classifiers 48

3.6 A New Algorithm: The Missing Step for NISQ Era....49
3.7 Quantum Hoeffding Tree Classifier....54
3.8 Summary....59
Chapter 4 - Results....60
4.1 Hypothesis Evaluation for Real Quantum Devices....60
4.2 Hypothesis Evaluation Utilizing Quantum Simulators....63
4.3 Bottleneck Analysis....65
4.4 New Quantum Hoeffding Tree Classifier Algorithm....67
4.5 Evaluation of Results....68
Chapter 5 - Discussion and Conclusions....71
5.1 Future Research....73
References....76

List of Figures

Figure 1: Botnet Architecture 4

Figure 2: Initial Experiment Design for Deployment 10

Figure 3: Botnet Components 14

Figure 4: Conjugate Gradient Algorithm (Nazareth, 2009) 30

Figure 5: Powell Algorithm (Brent, 2013) 34

Figure 6: Research Methodology 42

Figure 7: Stable Deployment Model after Bottleneck Analysis 48

Figure 8: Model Serving 49

Figure 9: Imagining Three Qubits in a Bloch Sphere for New Distance Function 49

Figure 10: Implementation of Hoeffding Tree Classifier (Tehrani et al., 2023) 55

Figure 11: Instrumentation of Hoeffding Tree Classifier for Online Learning (Tehrani et al., 2023) 56

Figure 12: Additional Code for Online Learning with Hoeffding Tree Classifier (Tehrani et al., 2003) 57

Figure 13: Novel Hybrid Quantum Binary Classifier Algorithm (Tehrani et al., 2023) 58

Figure 14: Deployment Hardening as Part of the Solution 66

List of Tables

Table 1: Domain Generation Algorithm Botnet Dataset Taken from Suryotrisongko and Musashi (2022) 8

Table 2: Maximum Accuracy of the Key Optimizer 39

Table 3: Descriptive Statistics (Tehrani et al., 2023)..... 45

Table 4: Accuracy and Standard Deviation of Different Classic Binary Classifiers 46

Table 5: Accessible Devices 47

Table 6: Actual Device Time and Accuracy (Tehrani et al., 2023) 60

Table 7: Classical Binary Classifiers as a Benchmark..... 61

Table 8: Comparison of Hybrid Quantum Binary Classifiers Versus Classical Binary Classifiers..... 61

Table 9: Results of Hybrid Quantum Binary Classifiers Over Simulators 64

Table 10: Quantum Hoeffding Tree Classifier Test Results (Tehrani et al., 2003)..... 68

Table 11: List of Classifiers Used..... 69

Chapter 1 - Introduction

1.1 Background

Quantum machine learning (QML) has attracted attention due to expected computational advantages in standard machine learning (ML) tasks. While many review papers have provided comprehensive overviews of the evolving landscape of QML research, a crucial challenge remains in achieving a concrete quantum advantage when implementing QML applications on existing noisy intermediate scale quantum (NISQ) devices.

This praxis explores hybrid quantum-classical approaches for binary classification algorithms, with the goal of enhancing performance through a combination of quantum and classical ML methods. Within the realm of QML algorithms, three distinct approaches can be identified:

1. Quantum adaptations of traditional ML algorithms that directly employ quantum computational paradigms.
2. Quantum-inspired ML algorithms that use principles of quantum computing to enhance classical computational methods.
3. Hybrid quantum-classical ML algorithms that strategically integrate both quantum and classical computational techniques to achieve optimized performance metrics.

By specifically focusing on the third approach, this research aims to make a significant contribution to the ongoing discussion on the practical applicability and effectiveness of QML algorithms, particularly in the context of contemporary NISQ devices.

This praxis explores the potential of Hybrid Quantum Binary Classification (HQBC) algorithms for application in cybersecurity analytics, specifically for addressing issues that

require binary decisions. HQBC algorithms leverage quantum circuits for data transformations or parameter calculations and have demonstrated improved accuracy in certain scenarios. Our main focus is on using HQBC to detect domain generation algorithm (DGA) botnets, a formidable cyber threat. The insights gained from this research are relevant for other binary decisions, such as spam, phishing, general botnets, traffic anomalies, insider/outsider threats, and legitimate user behavior. Additionally, this praxis provides valuable perspectives on the readiness of HQBCs in detecting emerging threats, such as the transformation of ransomware into spyware resulting from the Ukraine–Russia conflict in early 2023.

Botnets pose a significant and increasing threat to cybersecurity infrastructure. These malicious networks use interconnected malware, commonly referred to as bots, along with command and control (C&C) servers to coordinate and carry out sophisticated cyberattacks. One particularly harmful variation of this threat is the botnet based on the domain generation algorithm (DGA), which avoids detection by generating C&C server domains algorithmically and hiding its communication channels within domain name system (DNS) traffic. The urgency of addressing DGA-based botnets is amplified in the era of 5G and Beyond-5G networks, where the sheer number of interconnected devices, potentially reaching billions, creates a vast and vulnerable attack surface. Adversaries who exploit DGA-based botnets can cause significant financial harm to targeted businesses. Even at the lower end of the financial scale, distributed denial-of-service (DDoS) attacks facilitated by these botnets can result in costs ranging from $300,000 to $400,000 US per hour.

This praxis involves three main contributions. First, the paper evaluates the performance of all currently available HQBCs in Qiskit. This is done by using four simulators and three different quantum devices (IonQ, Quantinuum, and Rigetti). Second, it identifies obstacles and deficiencies in the current cloud-based access to quantum devices and proposes solutions to overcome these challenges. Third, the praxis proposes a new approach to achieve quantum advantage in the NISQ era of cybersecurity analytics.

1.2 Research Motivation

A botnet is a network of compromised computing devices that have been taken over by malicious software, commonly referred to as bots. These infected devices are controlled remotely by a person or entity known as the botmaster. The botmaster directs the network's activities through a centralized C&C infrastructure, as shown in Figure 1. Other possible architectures for implementing botnets include echo-based and peer-to-peer (P2P) models, which differ in how the compromised devices communicate and are controlled.

Botmasters exploit botnet networks to carry out various malicious activities, including but not limited to initiating DoS attacks, distributing spam or malware, extracting personal information, and engaging in espionage by collecting sensitive data from databases and information systems. Additionally, botnets can play a crucial role in executing targeted attacks against specific individuals or organizations.

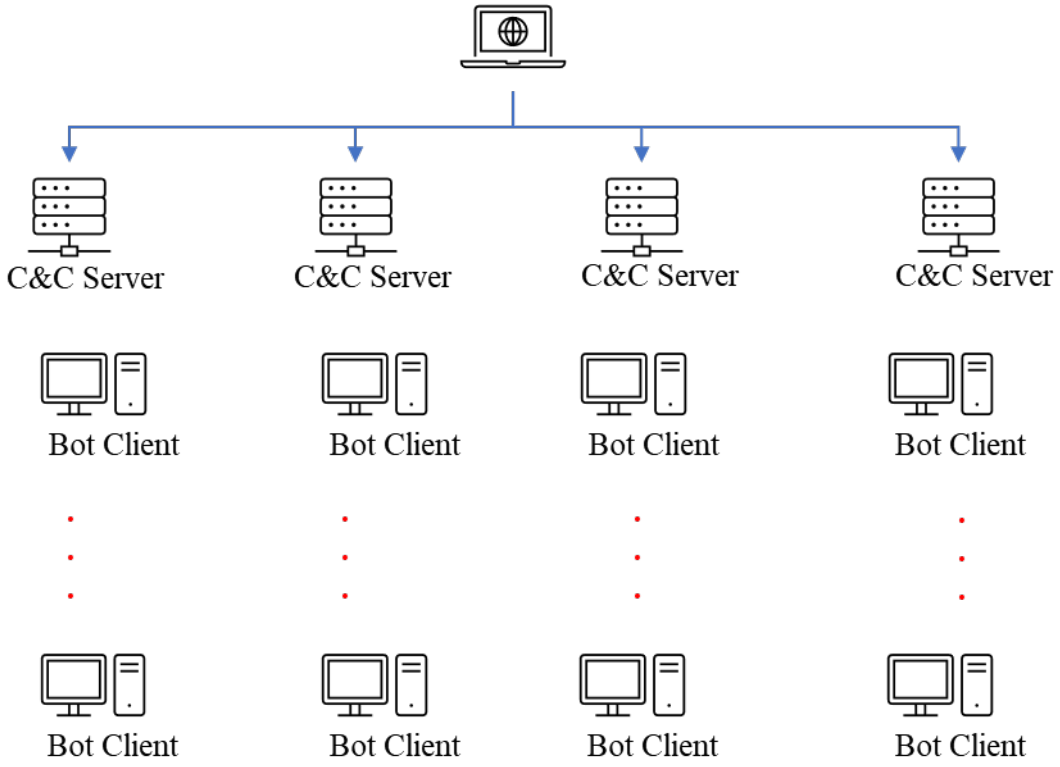

Figure 1: Botnet Architecture

A defining characteristic that sets botnets apart from other types of malware is the presence of a C&C infrastructure. This centralized system enables a remote botmaster to distribute commands and code updates to individual bots, instructing them to perform specific tasks. Bots can establish a connection with the C&C server either directly through its IP address or indirectly through a domain name. Connecting via a fixed IP address exposes the C&C server to inherent vulnerabilities, making it susceptible to detection and interception by firewalls and other security measures. To mitigate this risk, bots commonly use algorithmically generated domain names that are understood by both the bot and the C&C server. These algorithmically generated domains have unique characteristics that distinguish them from benign domain names, reducing the effectiveness of traditional detection methods.

1.3 Problem Statement

The use of DGA domains to conceal C&C enables botmasters and bots to avoid detection by security tools, allowing them to carry out unexpected attacks.

1.4 Thesis Statement

The integration of quantum binary classifiers with classical binary classifiers (CBCs) has the potential to improve the accuracy of detecting DGA-based botnets. This improvement can be achieved by utilizing various quantum hardware, such as IonQ, Quantinuum, and Rigetti.

1.5 Research Objectives

This praxis aims to achieve two objectives. Firstly, to evaluate the accuracy and speed of HQBCs compared to CBCs for the following algorithms:

1. Quantum support vector machines (QSVS)
2. Pegasos quantum support vector machines (PegasosQSVS)
3. Variational quantum classifier (VQC)
4. Quantum neural networks (QNNs)

Secondly, to conduct an empirical analysis to identify the advantages and disadvantages of using HQBCs and to provide practical solutions tailored for the NISQ era.

1.6 Research Questions and Hypotheses

The research questions in this praxis are the following:

1. How do HQBCs such as QSVC, PegasosQSVC, QNN, and VQC perform in terms of speed and accuracy compared to CBCs when applied to NISQ hardware, including both simulators and real devices?

2. Do HQBCs need to be adapted to ensure optimal functionality on NISQ platforms?

The research questions were addressed by proposing the following hypotheses:

- Null Hypothesis: Existing HQBCs can be readily used with NISQ hardware to achieve comparable or superior results to CBCs.
- Alternate Hypothesis: Existing HQBCs require adaptation to achieve comparable or superior results to CBCs when used with NISQ hardware.

1.7 Scope of Research

1.7.1 Algorithm

The research scope includes evaluating four different hybrid QML algorithms for detecting DGA-based botnets: QSVS, PegasosQSVS, VQC, and QNNs. Detailed explanations of how these algorithms were chosen will be provided in Chapters 2 and 3.

All four algorithms use quantum computation to enhance support vector machines (SVMs) in unique ways. SVMs are widely used and effective in classification problems. Typically, the training process for SVMs is formulated as a constrained quadratic programming problem. The main strategy is to minimize the unconstrained empirical loss function while also considering a penalty for the norm of the classifier being learned. Chapters 2 and 3 will provide comprehensive explanations of how these algorithms were selected and how they are used in the research.

Formally, considering a training set $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ and $y_i \in \{+1, -1\}$, SVMs aim to find the minimum of the following equation:

$$\min_{\boldsymbol{w}} \frac{\lambda}{2} \| \boldsymbol{w} \|^2 + \frac{1}{m} \sum_{(\boldsymbol{x},y)\in S} \ell(\boldsymbol{w}; (\boldsymbol{x}, y)) \qquad \text{Equation 1}$$

where

$$\ell(\boldsymbol{w};(\boldsymbol{x},y)) = max\{0,1 - y\langle \boldsymbol{w},\boldsymbol{x}\rangle\} \qquad \text{Equation 2}$$

In this notation, $\langle \boldsymbol{a},\boldsymbol{b}\rangle$ represents the standard inner product between the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. The objective function of the above is given by $f(\boldsymbol{w})$. The goal is to find an ϵ-accurate solution, $\hat{\boldsymbol{w}}$, where:

$$\ell(\boldsymbol{w};(\boldsymbol{x},y)) = max\{0,1 - y\langle \boldsymbol{w},\boldsymbol{x}\rangle\} \qquad \text{Equation 3}$$

Chapter 3 elaborates on how QSVC modifies the data format in the training set S and discusses how PegasosQSVS modifies the optimization method using a stochastic approach (Suryotrisongko, 2020). Furthermore, it details how both VQC and QNNs alter the SVM with quantum circuit parameter calculations, but in different ways.

1.7.2 Dataset

The research used two datasets for both training and evaluating the algorithms: the BOTNET DGA DATASET (Suryotrisongko, 2020) and UMUDGA (Zago et al., 2020). One of the datasets served as a ground-truth dataset, consisting of 1,803,333 domain names, as proposed in the BOTNET DGA DATASET. This dataset was created by using the top one million domains from Alexa and including the top 10 DGA families associated with botnets. The specific DGA families are outlined in Table 1.

Table 1: Domain Generation Algorithm Botnet Dataset Taken from Suryotrisongko and Musashi (2022)

Dataset name	Number	Sample Content
Alexa	1,000,000	google.com
Conficker	100,000	hkaswerxcnjf.biz
Cryptolocker	100,000	awersacpkhaj.net
Goz	1,667	mixrosoft.ru
Matsnu	100,000	sdfiusyduifysudapplyfitcple.com
New_Goz	1,666	Asioduiausdiu3238z03.org
Pushdo	100,000	sdkasjd.ru
Ramdo	100,000	asdjkajksdj.org
Rovnix	100,000	thebeautifuldomain.com
Tinba	100,000	aeaedsd.com
Zeus	100,000	asdasdasd 41j1.net

The data underwent preparation for analysis through the implementation of feature engineering, which transformed domain names (e.g., www.google.com) into seven distinct features:

- Character length of domain (CharLength): The count of characters in the domain name.
- Entropy score: Calculated using Shannon's formula, this score is based on the frequency distribution of characters within the domain name.
- Relative entropy (RE) compared to Alexa domains (REAlexa): This metric quantifies how the entropy of a domain compares to that of domains listed by Alexa.
- Lowest RE compared to botnet DGA domains (MinREBotnets): Representing the smallest RE when compared to domain names associated with botnet DGAs.
- Information radius measurement (IRad): A value indicating the information radius.
- New value calculated using decision tree algorithm (TreeNewFeature): A novel feature derived from the application of a decision tree algorithm.

- Reputation score based on Alexa (ReputationAlexa): This score reflects the reputation of the domain, as rated by Alexa.

Chapter 3 provides a thorough discussion of this dataset and the methodology used to create these features. The second dataset used for evaluation in this praxis is UMUDATA. It consists of over thirty million manually labeled generated domain names with an algorithmically derived feature set, specifically designed for ML. The dataset includes 50 of the most significant malware variants, available in both domain list and feature collection formats. More information about this dataset will be provided in Chapter 3.

1.7.3 Experiment

The experiment was conducted in the Azure Quantum cloud and Colab environment using the Rigetti, Quantinuum, and IonQ quantum hardware devices and their respective simulators (Figure 2, Table 5). The Qiskit simulator was used as a benchmark for comparison. Chapter 3 explains how the architecture below in Figure 2 and Figure 14, can be utilized in the NISQ era to bring stability to cloud-based quantum computing environments when a significant number of loops are required for QML.

1.7.1 Code Sharing

Repeating this experiment could result in significant costs due to the limitations of the NISQ era. The purpose of openly sharing the research source code is to reduce expenses for fellow researchers and facilitate future investigations that expand upon and surpass current models. To accomplish this, Tehrani et al. (2023) have made all the source code readily available on GitHub.

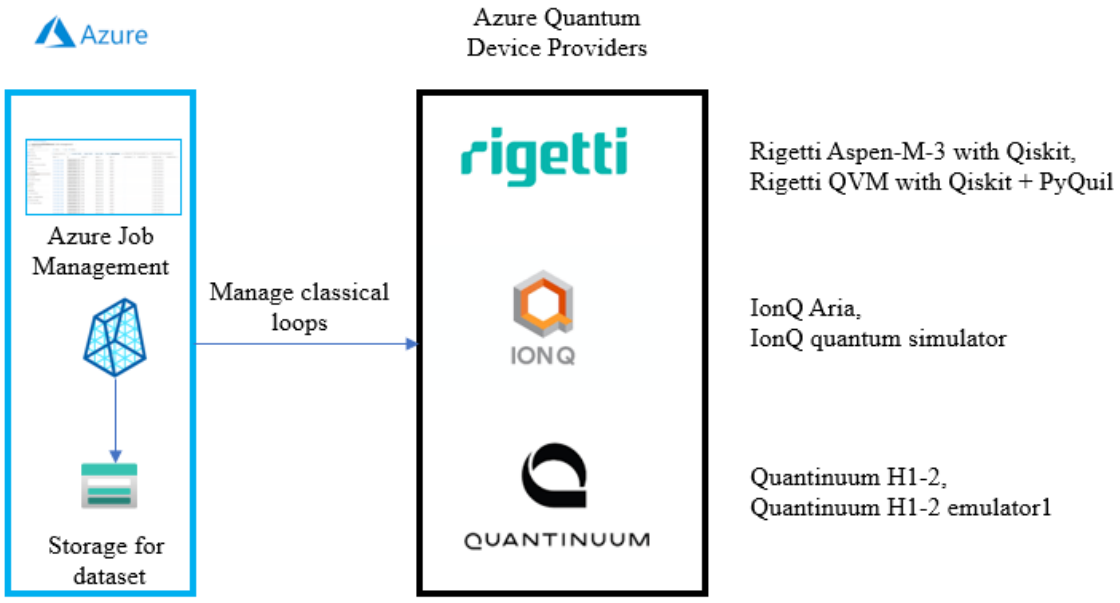

Figure 2: Initial Experiment Design for Deployment

1.7.2 Contribution

The sample size of the data in our QML research increased from 1,000 to 5,000 through simulations using real quantum devices and a novel Quantum Hoeffding Tree Classifier (QHTC). The algorithm demonstrated an average accuracy of 91.2%, with a perfect accuracy rate of 100% achieved in the final round. The total computational time was recorded as 1,687 seconds, which aligns with typical durations for local quantum simulations.

Building on existing literature, a series of HQBC algorithms were effectively executed on physical quantum devices. These algorithms processed up to 100 fixed data samples randomly, achieving a maximum accuracy rate of 59.0%.

In addition to these key results, this praxis also contributes the following findings:

- The vulnerabilities associated with the instability of prolonged code execution across three distinct Azure Quantum providers were mitigated by employing code hardening techniques.
- Instead of relying on traditional loop-based algorithms that are dependent on gradient descent, the batch-wise Hoeffding tree algorithm was used to enhance computational efficiency.
- A thorough comparison of various binary classifiers was conducted on actual devices, simulations based on real devices, and quantum simulators, all using the IEEE Botnet DGA dataset for consistency.

Furthermore, this research paves the way for the application of quantum cybersecurity analytics, introducing an innovative aspect to combat DGA botnets.

1.8 Research Limitations

The research has certain limitations that should be considered. Firstly, the leading quantum processors used in the experiment have approximately fifty to a few hundred qubits, which may not be enough for fault tolerance or achieving quantum supremacy. Secondly, quantum processors are affected by noise, which is caused by environmental factors, and quantum circuits or gates may lose their quantum state due to quantum decoherence. Additionally, in the current NISQ era, quantum processors do not have the sophistication for continuous quantum error correction. The term “intermediate scale” is used to describe the moderate number of qubits and gate fidelity in these quantum processors, which could impact the accuracy and performance of the experiment.

Another limitation is the cost of the experiment. NISQ machines are expensive, and the allocated budget of $15,000 US for real hardware may not be enough if the experiment needs to be repeated or if there is an incorrect implementation.

Furthermore, the research was limited by the current state of QML development. While many theoretical and experimental studies have explored QML algorithms, most have only used toy datasets. There have been relatively few experiments conducted on real devices and real datasets, which could affect the generalizability of the experiment's findings.

1.9 Organization of Praxis

This praxis compares the performance of CBCs and HQBCs in detecting DGA-based botnets. The justification for selecting HQBCs will be based on relevant foundational research outlined in Chapter 2. In Chapter 3, a mathematical framework will be presented to illustrate how HQBCs enhance SVMs through various mechanisms. The empirical findings will be thoroughly discussed, and proposed mitigation methods for observed barriers will be outlined in Chapter 4. The praxis will also examine the limitations of models under ideal conditions, comparing their accuracy and time on different hardware and simulators. These results are expected to lay the foundation for future methodologies that require less data to achieve quantum advantage. Chapter 5 will draw conclusions from the praxis, providing an evaluation of hypotheses based on the results and offering recommendations for Security Information and Event Management experts interested in utilizing quantum devices for cyber defense systems in the NISQ era.

Chapter 2 - Literature Review

2.1 Botnets

2.1.1 General Botnets

Botnets are networks of computers that are illicitly controlled by hackers (Kim et al., 2011). These networks can range from hundreds to millions of computers and are overseen by an individual known as a botmaster, who uses them to launch cyberattacks. The origins of botnets can be traced back to 1988-1989 when Greg Lindahl created the first botnet using a bot named GM that operated on the IRC protocol (Schiller et al., 2007). Initially designed to assist system administrators, botnets took a malicious turn in 1999 with the introduction of harmful code by Pretty Park, marking the beginning of their use for malicious purposes. Since then, botnets have evolved, employing sophisticated techniques that present challenges for security professionals (Tuptuk & Hailes, 2018).

A typical botnet consists of three main components: the botmaster, the C&C server, and a network of bots (Ullah et al., 2013). Typically, botnets use an IRC server for remote control. The botmaster controls the bots through the C&C server, which sends commands and updates. The bots, which are infected devices, perform various tasks such as hiding the system, receiving commands, and obtaining updates (see Figure 3). They primarily operate in two models—Client–Server and P2P—which facilitate the expansion and dissemination of bot networks.

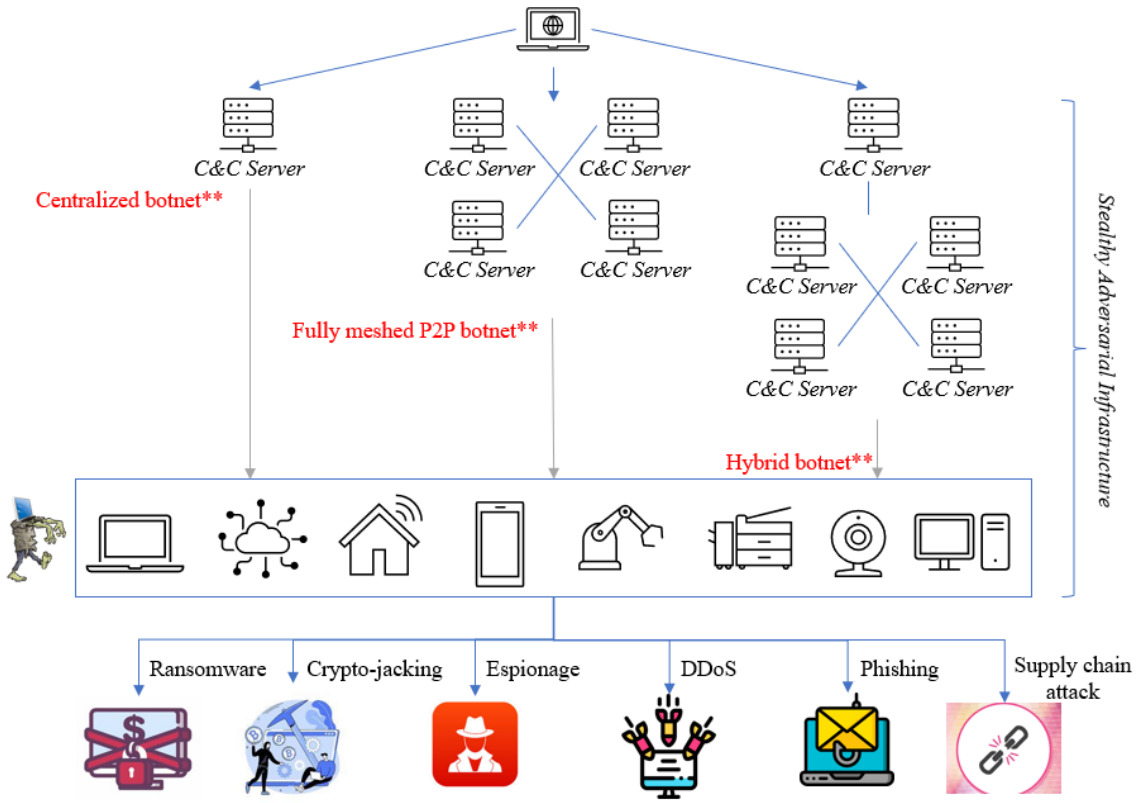

Figure 3: Botnet Components

Botnets typically operate covertly and spread widely, remaining dormant until the hacker issues commands through the C&C server. These networks serve various purposes:

- DoS attacks: Botnets can initiate attacks, causing significant damage, as observed in incidents targeting Amazon Web Services and the OVH company (Tuan et al., 2022).
- Spam messaging and email distribution: Botnets are used for sending spam, posing risks to users with limited security knowledge. They contribute to network traffic congestion, with a substantial portion of spam emails originating from botnets (Khan et al., 2015; John et al., 2009).

- Information theft and espionage: Botnets engage in stealing information and monitoring user activities, posing risks to government agencies and organizations (Ianelli & Hackworth, 2007).
- Automated internet behaviors: Botnets can mimic user actions online for financial gain (Chen, 2018).

In summary, botnets pose a significant threat to the internet, generating substantial profits for hackers and causing billions of dollars in global economic damage (Anderson et al., 2019; Putman et al., 2018). The relatively low cost of setting up and renting a botnet in relation to potential earnings highlights the crucial importance of detecting and preventing them in the field of cybersecurity.

2.1.2 Domain Generation Algorithm Botnets

DGA-based botnets function as clients within a client-server architecture. Once a computer has been infected, these bots connect to the C&C server to receive instructions. They typically obtain the IP address of the C&C server through DNS queries, using domain names that are automatically generated and frequently changed to evade security measures.

Botnets go through three main stages during their lifecycle:

1. Infection: During this initial stage, malicious codes and malware are distributed through messages, emails, or files. Computers infected by these codes become bots under the control of the C&C server.
2. Stealth: In this phase, bots interact with the botmaster through the C&C server to receive software updates and send gathered information. They avoid detection by security tools, such as antivirus software.

3. Attack: Bots carry out attacks based on commands received from the C&C server. These attacks can involve launching denial-of-service campaigns and spreading malware. This is also the stage where bots are most vulnerable to detection. Sometimes, the botmaster may remove unnecessary bots from the botnet.

Maintaining communication with the C&C server is crucial throughout these stages. However, using a fixed IP address for the C&C server would be too conspicuous and easily blocked by security systems. To overcome this, bots use domain names to secretly query the server's IP address. Early techniques involved using several domain names pointing to the same IP address (domain name technique) or multiple domain names pointing to multiple IP addresses (multihoming technique; Schiller et al., 2007). However, these methods were too simplistic, and the DGA was developed as a more sophisticated solution. In this approach, both C&C servers and bots use an agreed-upon algorithm to generate domain names that lead to an IP address specified by the botmaster. This method reduces the risk of detection and allows the C&C server to easily change its IP address while still being discoverable by the bots.

Identifying DGA botnets involves determining whether a domain name is malicious or benign. This process can develop into a classification task wherein the family of domain names associated with each botnet type is identified. Rapid detection of these queries is crucial in putting a stop to botnet operations, even after a device has been infected. Karim et al. (2014) identified two primary methods for detecting botnets:

1. Honeynet-based techniques: These involve setting up honeynets, also known as trap networks, to discreetly gather information about botnets and analyze their characteristics and behaviors. Honeynet-based detection systems are relatively easy

to establish and do not require excessive system resources. However, they often face scalability limitations.

2. Intrusion detection system (IDS)-based techniques: This method utilizes an IDS, which can be either a hardware device or a software application. IDSs typically operate by monitoring and analyzing network packets to detect signs of botnets. They monitor system services to identify malicious activities or violations of a security policy, and then alert management. There are several types of IDS-based techniques, including:
    - Anomaly-based detection, which identifies unusual behavior patterns.
    - Signature-based detection, which uses known patterns to find botnets.
    - DNS-based detection, focusing on identifying malicious DNS connections.

This praxis specifically focuses on DNS-based botnet detection. This approach is beneficial because it can prevent botnets from contacting their botmaster through C&C servers, effectively disabling them even after infection. DNS analysis also requires fewer computing resources compared to other methods.

Additionally, this praxis explores the use of ML for anomaly detection in identifying botnets. It evaluates current machine learning techniques, especially supervised learning, and compares them with QML algorithms. This approach aims to improve the efficiency of botnet detection.

2.2 Quantum Machine Learning

In Machine Learning, detecting DGA-based botnets is a classification task, wherein data is categorized into a set of classes labeled C = {1, 2, ..., c}. This task comprises two sets of data: a training set T and a test set S. Both sets are labeled by an unknown function

m, which maps elements from the union of T and S to the class set C. However, this function m is concealed from the programmer.

The programmer has access to both data sets, S and T, but only the labels of the training set T are known. The known part of the function m, which maps elements of T to their corresponding classes in C, is denoted as $m_|T$. This can be expressed as $m_|T: T \rightarrow C$, such that $m_|T(t) = m(t)$ for every element t in T.

The objective of the programmer is to leverage knowledge of $m_|T$ to create a new mapping function for the test set S, denoted as m~. This new function $m\sim: S \rightarrow C$ should predict the correct class for each element in S with high accuracy. The success rate of this prediction is measured by how many elements in S are correctly classified, represented as:

$$\nu_{succ} = \frac{|\{s \in S \mid m(s) = m\sim(s)\}|}{|S|} \qquad \text{Equation 4}$$

For this learning task to have meaningful implications, it is presumed that there is a similarity in how the function m behaves across both the training set T and the test set S. It is believed that both sets are drawn from a family of d-dimensional distributions $\{p_c: \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}\}_{c \in C}$, and their labels are assigned based on these distributions. The goal is to learn the theoretical classification function m, which is assumed to be derived in this way. However, the programmer only has a large but limited number of samples and their labels, without direct access to the distributions and the labeling function.

Traditionally, to address this problem, a family of functions m~: ⟨θ, S⟩ → C is created. These functions are parameterized by a set of parameters θ, which are determined from m_|T through an optimization process on a classical cost function. Additionally, quantum

algorithms are considered for generating some or all of the classification protocol m on a quantum computer.

2.2.1 Quantum Support Vector Machines

QSVMs involve the following five steps (Rebentrost et al., 2014):

1. Prepare quantum states: Given quantum vectors $|x_j\rangle = \frac{1}{|x_j|}\sum_{k=1}^{N}(x_j)_k|k\rangle$, norms $|x_j|$, and labels y_j, quantum RAM constructs states using $O(MN)$ hardware resources and $O(log\ MN)$ operations.
2. Kernel matrix construction: Construct a kernel matrix K with $K_{ij} = x_i^T \cdot x_j$. This step is essential for the dual SVM formulation.
3. Quantum least-squares SVM: Reformulate SVM as a least-squares problem. Replace inequalities with equalities using slack variables e_j:

$$y_j(w \cdot x_j + b) \geq 1 \rightarrow (w \cdot x_j + b) = y_j - y_j e_j \qquad \text{Equation 5}$$

Solve linear system:

$$F\begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix} \qquad \text{Equation 6}$$

where

$$F = \begin{bmatrix} 0 & 1^T \\ 1 & K + \gamma^{-1}I \end{bmatrix} \qquad \text{Equation 7}$$

4. Solve linear system with quantum matrix inversion: Use quantum matrix inversion to solve $\hat{F}|b\alpha\rangle = |y\rangle$, where $\hat{F} = F/tr\ (F)$ and $\| F \| \leq 1$.

5. Classification: Classify new query state $|x\rangle$. Construct states $|\tilde{u}\rangle$ and $|\tilde{x}\rangle$ and perform a swap test to classify $|x\rangle$. The success probability of the swap test determines the classification.

The complexity of QSVM is significantly influenced by the efficiency of quantum matrix inversion and kernel estimation. Quantum matrix inversion using phase estimation operates in $O(log(M) * \frac{\delta^2}{\varepsilon})$, where M is the size of the kernel matrix, δ is the condition number of F, and ε is the desired precision. On the other hand, the complexity of the kernel estimation step depends on the particular quantum circuits employed for inner product estimation and may vary based on the chosen kernel function.

2.2.2 PegasosQSVC

PegasosQSVC employs stochastic gradient descent on the primal objective, using a carefully selected step size (Shalev-Shwartz et al., 2011). In each iteration, PegasosQSVC follows this procedure: Initially, $\boldsymbol{w}_1$ is set to the zero vector. On the t-th iteration of the algorithm, a random training sample $(\boldsymbol{x}_{i_t}, y_{i_t})$ is selected through uniform random selection of a random index $i_t \in \{1, \dots, m\}$. Using an estimation derived from the training example $(\boldsymbol{x}_{i_t}, y_{i_t})$ as the objective results in the following equation:

$$f(\boldsymbol{w}; i_t) = \frac{\lambda}{2} \| \boldsymbol{w} \|^2 + \ell\left(\boldsymbol{w}; (\boldsymbol{x}_{i_t}, y_{i_t})\right) \qquad \text{Equation 8}$$

The sub-gradient of the approximated objective is expressed as follows:

$$\nabla_t = \lambda \boldsymbol{w}_t - \mathbb{1}[y_{i_t} \langle \boldsymbol{w}_t, \boldsymbol{x}_{i_t} \rangle < 1] y_{i_t} \boldsymbol{x}_{i_t} \qquad \text{Equation 9}$$

where $\mathbb{1}[y\langle \boldsymbol{w}, \boldsymbol{x}\rangle < 1]$ is the indicator function that assumes a value of one when its condition is met (i.e., when **w** results in a non-zero loss for the example$(\boldsymbol{x}, y)$) and is zero otherwise. Following this, an update is proceeded with:

INPUT: S, λ, T

INITIALIZE: Set $\boldsymbol{w}_1 = 0$

FOR $t = 1,2, \ldots, T$

Choose random $i_t \in \{1, \ldots, |S|\}$ uniformly.

Set $\eta_t = \frac{1}{\lambda t}$

If $y_{i_t}\langle \boldsymbol{w}_t, \boldsymbol{x}_{i_t}\rangle < 1$, then:

Set $\boldsymbol{w}_{t+1} \leftarrow (1 - \eta_t\lambda)\boldsymbol{w}_t + \eta_t y_{i_t}\boldsymbol{x}_{i_t}$

Else (if $y_{i_t}\langle \boldsymbol{w}_t, \boldsymbol{x}_{i_t}\rangle \geq 1$):

Set $\boldsymbol{w}_{t+1} \leftarrow (1 - \eta_t\lambda)\boldsymbol{w}_t$

[Optional: $\boldsymbol{w}_{t+1} \leftarrow min\left\{1, \frac{1/\sqrt{\lambda}}{\|\boldsymbol{w}_{t+1}\|}\right\}\boldsymbol{w}_{t+1}\Big]$

OUTPUT: $\boldsymbol{w}_{T+1}$

$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta_t\nabla_t$ using a step size of $\eta_t = 1/(\lambda t)$

It is important to note that this update can be reformulated as follows:

$$\boldsymbol{w}_{t+1} \leftarrow \left(1 - \frac{1}{t}\right)\boldsymbol{w}_t + \eta_t\mathbb{1}\left[y_{i_t}\langle \boldsymbol{w}_t, \boldsymbol{x}_{i_t}\rangle < 1\right]y_{i_t}\boldsymbol{x}_{i_t} \qquad \text{Equation 10}$$

After a predetermined number of iterations, T, the last iterate, $\boldsymbol{w}_{T+1}$, is produced.

2.2.3 Quantum Neural Network

Schuld et al. (2014) proposed a QNN with the following components:

Initialization: Define "qurons" as qubits in a 2^N-dimensional Hilbert space, where each quron represents a neuron's state (active or resting). Initialize the state of the network as:

$$|\psi\rangle = \sum_{i=1}^{2^N} a_i |x_1 x_2 \dots x_N\rangle_i \qquad \text{Equation 11}$$

Here, $|\psi\rangle$ is the quantum state of the network, a_i are complex amplitudes, and $|x_1 x_2 \dots x_N\rangle_i$ are the basis states.

Quantum neural dynamics: Use quantum mechanical principles such as superposition, entanglement, and interference to evolve the network. Address the challenge of integrating nonlinear neural dynamics with linear quantum dynamics.

Quantum measurement for activation: Employ quantum measurement to simulate the activation function of neurons. This process maps the quantum state of qurons (quantum neurons) to a classical output, similar to an activation function in classical neural networks.

Output processing: Process the output state of the network after measurement, which encodes the result of the quantum neural computation. This step involves interpreting the measurement results to determine the final output of the QNN.

Stabilization and dynamics modeling: Investigate the use of Open Quantum Walks or similar models to simulate stable state dynamics in the QNN. Focus on achieving a model that can replicate the attractor-like dynamics of classical neural networks within the quantum framework.

Farhi and Neven (2018) proposed a simplified model of QNN for NISQ devices, consisting of three stages: setup, representation, and learning.

A. Setup of QNN for NISQ devices:

- Step 1: Introduction to the problem – Classify a dataset of strings, each with a binary label, using a QNN.
- Step 2: Setting up the QNN – Use a dataset of strings z= $z_1 z_2 \ldots z_n$ with binary labels $l(z)$. The quantum processor operates on $n + 1$ qubits.
- Step 3: Implementing unitary transformations – Use basic unitaries $\{U_a(\theta)\}$, combining them into $U(\theta) = U_L(\theta_L) \ldots U_1(\theta_1)$.
- Step 4: Processing and measuring the input state – Transform input state $|z1\rangle$ by $U(\theta)$. Measure σ_y (or Y_{n+1}) on the readout qubit.
- Step 5: Objective and loss function – Minimize the loss function:

$$loss(\theta, z) = 1 - l(z)\langle z1|U^{\dagger}(\theta)Y_{n+1}U(\theta)|z1\rangle \qquad \text{Equation 12}$$

- Step 6: Training the QNN – Start with random or inspired θ. Adjust θ to reduce loss for training examples, possibly using gradient descent.

B. Representation of QNN

- Step 1: Representing subset parity and majority:
  - Subset parity: $PS(z) = \oplus_j a_j b_j$ for a subset S.
  - Subset majority: $MS(z) = sign\left(\sum a_j z_j\right)$.
- Step 2: Quantum measurement for label prediction – Predict label value, determined by the expectation of $sin\ \beta \sum a_j Z_j$ where $\beta = 0.9\pi/n$.

- Step 3: Classifying handwritten digits (MNIST example) – Apply the quantum network to a downsampled version of the MNIST dataset, focusing on distinguishing digits 3 and 6 from a set of 6,031 samples.
- Step 4: Batch learning with quantum states – Present classical data in quantum superposition states for batch learning. States for +1 and -1 labels are $|\pm 1\rangle = N_{\pm}\sum_{z:l(z)=\pm 1} e^{i\phi_z}|z1\rangle$.

C. Learning

- Step 1: Learning subset parity – Efficiently learn via stochastic gradient descent, using U() for subset parity with $_j = \frac{\pi}{2}$ for bits in the subset and $_j = 0$ otherwise.
- Step 2: Learning subset majority – For subset majority, set $= \frac{0.9\pi}{n}$ in $U()$, with $_j = 1$ for bits in the subset. Efficiently train to represent the subset majority.
- Step 3: Learning to distinguish digits (MNIST Example) – Apply QNN to downsampled MNIST data. Learned to classify real-world data with low categorical error.
- Step 4: Classical data presented in superposition – Present data as quantum states in superpositions for batch learning and evaluate the gradient on a loss function for these states.
- Step 5: Learning the properties of quantum states – Classify labels based on properties of quantum states. Use a Hamiltonian with positive and negative eigenvalues to label states, achieving high accuracy in classifying quantum states.

2.2.4 Variational Quantum Classifier Algorithm

Havlíček et al. (2019) introduced a promising approach known as VQC, which has gained substantial interest from the research community. The VQC process involves the following steps:

1. Data encoding into quantum states:
   - Input data: $x \in \Omega \subseteq R^n$.
   - Feature map: Encode data into a quantum state using a feature map, $U_\Phi(x)$, applied to a reference state $|0^n\rangle$.
   - Quantum state: The resulting quantum state is $|\Phi(x)\rangle = U_\Phi(x)|0^n\rangle$.
2. Variational quantum circuit:
   - Parameterized circuit: Apply a variational quantum circuit $W(\theta)$ to the quantum state. This circuit is parameterized by $\theta \in R^{2n(l+1)}$, where l is the number of layers in the circuit.
   - Resulting state: The quantum state after applying $W(\theta)$ is $W(\theta)|\Phi(x)\rangle$.
3. Binary measurement:
   - Measurement operator: For a binary classification task with labels $y \in \{+1, -1\}$, define a binary measurement operator $M_y = \frac{1}{2}(1 + yf)$, where f is a chosen Boolean function mapping bit strings to $\{+1, -1\}$.
   - Probability of outcome: The probability of obtaining outcome y given input x is $p_y(x) = \langle \Phi(x)|W^\dagger(\theta)M_y W(\theta)|\Phi(x)\rangle$.
4. Empirical distribution and decision rule:
   - Measurement shots: Perform R repeated measurement shots to estimate the empirical distribution $\hat{p}_y(x)$.

- Label assignment: Assign the label $\tilde{m}(x) = y$ whenever $\hat{p}_y(x) > \hat{p}_{-y}(x) - yb$, with an additional bias parameter $b \in [-1,1]$.

5. Training and optimization:
   - Empirical risk: Define the empirical risk $R_{emp}(\theta)$ as the error probability of assigning the incorrect label averaged over the training set.
   - Optimization: Adjust parameters θ (and b if included) by minimizing $R_{emp}(\theta)$ using an optimization algorithm (e.g., stochastic gradient descent).
6. Classification phase:
   - Fixed parameters: After training, use the optimized parameters θ^* for classification of new, unlabeled data.
   - Output: The label for a new data point s is determined using the decision rule with the optimized parameters.

2.2.5 Qiskit Approach to Hybrid Quantum Binary Classification

Qiskit, a quantum computing software development framework developed by IBM, has become an essential tool in the field of quantum computing (Qiskit 0.7.0, 2023b). It provides a modular and interactive platform for designing quantum algorithms, simulating quantum circuits, and conducting experiments on IBM's quantum computers. With a focus on accessibility and education, Qiskit offers a comprehensive library of quantum operations and algorithms. The framework is structured into Terra for creating quantum circuits, AER for running simulations, Ignis for quantum error correction, and Aqua for quantum algorithms, enabling a wide range of quantum computing tasks. Qiskit's versatility and user-friendly interface significantly contribute to democratizing quantum

computing, allowing a broader audience to explore and contribute to this rapidly evolving field.

In the Qiskit 0.7.0 (2023b) framework, there is a notable demonstration of various implementations of QNNs within the Qiskit-machine-learning package. These implementations combine principles of quantum computing with conventional neural network (NN) methodologies, providing a unique perspective on machine learning. Classical NNs are algorithmic models inspired by the human brain, organized in layers of interconnected nodes. QNNs combine these NNs with parametrized quantum circuits, using quantum states and gates to process data. The Qiskit tutorial explains this integration, highlighting both the ML and quantum computing aspects of QNNs. Qiskit's machine learning module introduces two implementations of QNN: EstimatorQNN, based on quantum mechanical observables, and SamplerQNN, based on quantum circuit measurement samples. These networks, using Qiskit primitives, can operate on both simulators and real quantum hardware. It is important to note that they are "stateless," lacking inherent training capabilities or weight storage, and are instead managed by specific algorithms or applications.

Qiskit 0.7.0 (2023a) presents a comprehensive guide on leveraging quantum computing to enhance traditional ML classification methods, particularly SVMs. Classical SVMs, which are crucial in pattern recognition and data mining, rely on kernels—collections of inner products in a transformed feature space—for classification. The tutorial addresses a critical scenario where classical kernel computation becomes inefficient, requiring exponentially scaling computational resources. It demonstrates how quantum processors can directly estimate the kernel in the feature space through supervised learning, involving

a training phase (kernel calculation and support vector determination) and a test/classification phase (classifying new unlabeled data). This approach highlights the integration of quantum computing capabilities with established ML techniques, potentially providing more efficient solutions to complex classification problems.

In the Qiskit 0.7.0 (2023b) software, an alternative to the standard QSVC is introduced. This alternative implements the PegasosQSVC algorithm, a primal estimated sub-gradient solver for SVMs, as detailed in the work of Shalev-Shwartz et al. The PegasosQSVC algorithm operates differently from the dual optimization approach used in the scikit-learn package. It utilizes the kernel trick and offers a training complexity that does not depend on the size of the training set. Consequently, PegasosQSVC is designed to train faster than QSVC for large datasets and can be used as a direct replacement for QSVC with appropriate hyper-parameterization.

Qiskit 0.7.0 (2023c) introduces a variational method for quantum classification within its software framework. This approach is different from the kernel-based approach of the QSVM. The VQC algorithm is part of the broader field of quantum-enhanced ML, which aims to combine quantum computing principles with conventional ML techniques. The tutorial associated with this software, along with the QSVM notebook, aligns with the main theme of supervised learning using quantum-enhanced feature spaces, as explored in referenced academic papers. The introduction of VQC represents a significant advancement in leveraging quantum computing for complex classification tasks, demonstrating the potential of quantum algorithms in improving traditional ML methods.

In this praxis, the Qiskit implementation was used for all HQBC algorithms with various optimizers, and therefore, a review of the optimizer used in QML is also provided.

2.3 Optimizers in Quantum Machine Learning

Optimizers play a crucial role as the primary architectural element affecting performance. In this section, we review their mathematical models in alphabetical order. Table 2 provides a summary of the best-reported performances for DGA-based botnet detection associated with each optimizer.

2.3.1 Analytical Quantum Gradient Descent

Analytical quantum gradient descent (AGQD) presents a new optimization method where gradients are calculated analytically directly on quantum hardware, as outlined by Singh et al. (2022). Unlike traditional gradient-based optimizers, AGQD relies on quantum hardware for gradient computation, making it an optimizer specific to quantum hardware. In AGQD, gradients of the cost function are determined for circuits consisting of specific parameterized gates. These gates, which have two eigenvalues, are represented as:

$$\mathcal{G}(\mu) = e^{-i\mu G} \qquad \text{Equation 13}$$

where $\mathcal{G}$ is a Hermitian operator. Using a parameterized gate serving as an ansatz, the cost function's gradient f for an operator Q can be computed using the parameter-shift rule. This rule involves a shift s and eigenvalue r, resulting in the following gradient calculation:

$$\begin{aligned} \partial_\mu f = \; & r\big(\langle\psi|\mathcal{G}^+(\mu+s)\hat{Q}\mathcal{G}(\mu+s)|\psi\rangle \\ & -\langle\psi|\mathcal{G}^+(\mu-s)\hat{Q}\mathcal{G}(\mu-s)|\psi\rangle\big) \\ = \; & r(f(\mu+s)-f(\mu-s)). \end{aligned} \qquad \text{Equation 14}$$

When multiple gates share a parameter μ, the gradient can be computed using the product rule. This involves independently adjusting the associated parameters in each gate and then aggregating the outcomes of these individual shifts.

In a design featuring a low-depth circuit where the generator $\mathcal{G}$, $\mathcal{G} \in \frac{1}{2}\{\sigma_x, \sigma_y, \sigma_z\}$, is a single qubit rotation gate, then $r = \frac{1}{2}$, and $s = \frac{\pi}{2}$ $(Mitarai\ et\ al., 2018)$.

2.3.2 Conjugate Gradient

The conjugate gradient (CG) method is an efficient optimization technique developed to solve systems of linear equations that have symmetric and positive-definite matrices. It starts with an initial guess, or it is set to zero if no guess is available, and iteratively produces a sequence of better approximate solutions for the problem. Each guess is derived from the previous one. CG is often used for unconstrained optimization problems, especially in energy minimization scenarios. Figure 4 (Nazareth, 2009) shows a commonly used version of the CG algorithm.

Algorithm 1 Conjugate Gradient

Require: Consider a simple system of equations, $\mathbf{Ax} = \mathbf{b}$, and an initial guess, $\mathbf{x}_0$.
Ensure: $\mathbf{p}_0 = \mathbf{r}_0$ and $k = 0$.
  $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$.
  while $\mathbf{r}_k$ is not sufficiently small **do**
    $\alpha_k := \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T A \mathbf{p}_k}$
    $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{p}_k$
    $\beta_k := \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T A \mathbf{r}_k}$
    $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
    $k := k + 1$
  end while{return $\mathbf{x}_k$ as the result}

Figure 4: Conjugate Gradient Algorithm (Nazareth, 2009)

2.3.3 Constrained Optimization by Linear Approximation

Constrained optimization by linear approximation (COBYLA) is a nonlinear, derivative-free optimization algorithm that relies on linear approximation techniques (Singh et al., 2022). The function to be optimized is denoted by $F(x)$. COBYLA evaluates

the function at $n+1$ vertices $(0,1,2,\cdots,n)$ of a simplex, which is an n-dimensional geometric object. The edges of the simplex are then interpolated using a linear polynomial $L(x)$, defined for all real numbers in the set R. Each iteration of COBYLA requires a trust region radius $(\Delta > 0)$ that can be automatically adjusted (Powell, 1994). The next set of variables is obtained by minimizing the function $L(x)$, subject to the following constraint (Singh et al., 2022):

$$|x - x_o| \leq \Delta \qquad \text{Equation 15}$$

Here, $F(x_o)$ represents the minimum value of $F(x)$ found thus far. In the case of unconstrained COBYLA, the following expression is used to determine the value of x:

$$x = x_o - \left(\frac{\Delta}{|\nabla L|}\right)\nabla L \qquad \text{Equation 16}$$

After evaluating $F(x)$, the next simplex is formed by replacing one of the vertices of the previous simplex with the new value of x.

2.3.4 Limited Memory Broyden–Fletcher–Goldfarb–Shanno-Bound Optimizer

Broyden–Fletcher–Goldfarb–Shanno (BFGS; Fletcher, 2000) is a quasi-Newton method that approximates the Hessian matrix, i.e., a matrix of second-order partial derivatives of a function, to make progress in the optimization process. L_BFGS_B stands for limited-memory BFGS-bound optimizer. Utilizing a limited amount of memory, it provides an approximation to the original BFGS algorithm (Byrd et al., 1995). The problem can be mathematically stated as follows:

$$minf(x) \qquad \text{Equation 17}$$

subject to

$$l \leq x \leq u \qquad \text{Equation 18}$$

The algorithm constructs a quadratic model of the function f for each iteration x_k. It considers the current function value f_k, the gradient g_k, and the positive limited memory approximation B_k, as follows:

$$m_k(x) = f(x_k) + g_k^T(x - x_k) + \cdots \frac{1}{2}(x - x_k)^T B_k(x - x_k) \qquad \text{Equation 19}$$

The algorithm employs the gradient projection method to approximately minimize m_k while adhering to the boundary conditions. This leads to a set of active bounds, which are treated as equality constraints during the minimization process.

2.3.5 Nelder–Mead Algorithm

The Nelder–Mead algorithm (Nelder & Mead, 1965) is a classic algorithmic method designed for finding the minimum of a multidimensional, unconstrained function without needing to calculate its derivatives. The algorithm seeks to minimize a given nonlinear function:

$$f: R^n \rightarrow R \qquad \text{Equation 20}$$

It starts with a simplex, a geometric shape with $n + 1$ vertices, and iteratively improves the positions of these vertices until the function is minimized. The algorithm relies on four parameters controlling its transformations: α, β, γ, and δ, representing reflection, contraction, expansion, and shrinkage, respectively. These parameters must adhere to certain conditions for the algorithm to function correctly, specifically: $\alpha \geq 0, 0 \leq \beta \leq 1$, $\gamma \geq 1, \gamma \geq \alpha$, and $0 \leq \delta \leq 1$.

2.3.6 Powell Algorithm

The Powell algorithm (Brent, 2013) is a method designed for solving unconstrained optimization problems, notable for not requiring the function to be differentiable, though it does need the function to be real-valued. The algorithm commences with an initial point and a set of directional vectors called the initial search vectors (Singh et al., 2022). The Powell optimization algorithm involves performing a one-dimensional minimization along specific vectors. This algorithm operates under the assumption that the target function, subject to optimization, follows a particular form:

$$f(\vec{x}) = \vec{x}^T A\vec{x} - 2\vec{b}^T\vec{x} + c \qquad \text{Equation 21}$$

where A is positive, definite, and symmetric, and $b \in R^n$. It has been proven that if $\vec{u}_1, \cdots, \vec{u}_m$ is a set of non-zero conjugate directions, then the minimum of $f(\vec{x})$ in the space spanned by $\vec{u}_1, \cdots, \vec{u}_m$ can be found at the point $\sum_{i=1}^{m} \beta_i \vec{u}_i$, where

$$\beta_i = \frac{\vec{u}_i^T \vec{b}}{\vec{u}_i^T A \vec{u}_i}. \qquad \text{Equation 22}$$

Let $\vec{x}_0$ be the initial point, and $\vec{u}_1, \cdots, \vec{u}_n$ be the set of conjugate vectors, then the Powell algorithm is as shown in Figure 5.

Algorithm 2 Powell Algorithm

Initialize $\vec{x_0}$ and $\vec{u_1}, \cdots, \vec{u_n}$
repeat
  for $i \leftarrow 1$ to n **do**
    Compute β_i to minimize $f(\vec{x}_{i-1} + \beta_i \vec{u}_i)$
    Set $\vec{x}_i \leftarrow \vec{x}_{i-1} + \beta_i \vec{u}_i$
  end for
  for $i \leftarrow 1$ to $n - 1$ **do**
    Replace $\vec{u}_i$ with $\vec{u}_{i+1}$
  end for
  Replace $\vec{u}_n$ with $\vec{x}_n - \vec{x}_0$
  Compute β to minimize $f(\vec{x}_0 + \beta \vec{u}_n)$
  Replace $\vec{x}_0$ with $\vec{x}_0 + \beta \vec{u}_n$
until convergence criteria is met

Figure 5: Powell Algorithm (Brent, 2013)

2.3.7 Sequential Least-Squares Programming Optimizer

The sequential least-squares programming (SLSQP) optimizer is a method used for optimizing a multi-variable function with the goal of minimizing it while adhering to specific bounds, equality constraints, and inequality constraints (Kraft, 1988). Here, a local sequential programming variant is presented, which is less complex compared to the original SLSQP method (Nocedal & Wright, 2006).

Consider the problem:

$$\underset{x}{minimize} f(x) \text{ subject to } c(x) = 0 \qquad \text{Equation 23}$$

In this equality-constrained problem, $f: \mathbb{R}^n \to \mathbb{R}$ and $c: \mathbb{R}^n \to \mathbb{R}^m$ are smooth functions. The algorithm primarily focuses on formulating the given problem $f(x)$ at the current iteration x_k into a quadratic programming sub-problem, which is then minimized to establish a new iteration x_{k+1}. In this context, the Lagrangian function is represented as $L(x, \lambda) = f(x) - \lambda^T c(x)$, and the Jacobian matrix of the constraints is indicated by:

$$A(x)^T = [\nabla c_1(x), \nabla c_2(x), \cdots, \nabla c_m(x)] \qquad \text{Equation 24}$$

where $c_i(x)$ is the i-th component of the vector $c(x)$. Considering these assumptions, the problem can be modeled using the quadratic program:

$$\begin{aligned} &\underset{p}{minimize}\ f_k + \nabla f_k^T p + \cdots \\ &\frac{1}{2} p^T \nabla_{xx}^2 L_k p \ \text{ subject to } \ A_k p + c_k = 0 \end{aligned} \qquad \text{Equation 25}$$

Assuming the Jacobian matrix $A(x)$ has full row rank, this problem has a unique solution (p_k, l_k) satisfying:

$$\nabla_{xx}^2 L_k p_k + \nabla f_k^T - A_k^T l_k = 0 A_k p_k + c_k = 0 \qquad \text{Equation 26}$$

which results in:

$$\begin{pmatrix} \nabla_{xx}^2 L_k & -A_k^T & A_k \\ 0 & & \end{pmatrix} (p_k \lambda_{k+1}) = (-\nabla f_k - c_k) \qquad \text{Equation 27}$$

Therefore, by non-singularity, $\lambda_{k+1} = l_k$; p_k solves the quadratic problem, and a new iterate (x_{k+1}, λ_{k+1}) can be generated.

2.3.8 Simultaneous Perturbation Stochastic Approximation

The simultaneous perturbation stochastic approximation (SPSA) is an optimization technique renowned for its efficiency in handling systems with numerous undetermined parameters (Spall, 1998). It excels in optimizing high-dimensional and large-scale problems (Fletcher, 2000). A key feature of SPSA is its utilization of stochastic gradient approximations, requiring only two measurements of the objective function (loss function), irrespective of the problem's size or dimensionality. This significantly reduces the

computational expense associated with optimization, especially in problems with a multitude of variables.

The general form of the SPSA optimization procedure is given by the following equation:

$$\hat{\theta}_{k+1} = \hat{\theta}_k - a_k \hat{g}_k(\hat{\theta}_k) \quad \text{Equation 28}$$

Where $\hat{g}k(\hat{\theta}k)$ is an estimate of the gradient of the loss function $g(\theta) = \frac{\partial L}{\partial \theta}$ with respect to the n-dimensional vector of variables θ. The gradient can be approximated using one-sided or two-sided perturbations of the variables denoted by $\hat{\theta}_k$. In the case of two-sided simultaneous perturbation, the gradient approximation takes the form:

$$\hat{g}_{ki}(\hat{\theta}_k) = \frac{\hat{y}(\hat{\theta}_k + c_k \Delta_k) - \hat{y}(\hat{\theta}_k - c_k \Delta_k)}{2 c_k \Delta_{ki}} \quad \text{Equation 29}$$

where $\Delta_k = (\Delta_{k1}, \Delta_{k2}, \cdots, \Delta_{kp})^T$ is a p-dimensional random perturbation vector specified by the user.

2.3.9 Truncated Newton Algorithm

The truncated Newton algorithm (TNC), also known as the Newton conjugate-gradient method, is designed for solving constrained minimization problems. Similar to Newton's method (Byrd et al., 1995; Grippo et al., 1989), the algorithm is Hessian-free and approximates the solution to the optimization problem:

$$minf(x) \quad \text{Equation 30}$$

Newton's method iteratively calculates the next iteration as

$$x_{k+1} = x_k + p_k \quad \text{Equation 31}$$

where

$$\nabla^2 f(x_k)p = -\nabla f(x_k) \qquad \text{Equation 32}$$

However, Newton's method for optimization may not always converge and can be computationally expensive. To address these issues, the TNC algorithm was introduced. TNC iteratively determines the Hessian and provides an approximate solution to the optimization problem, similar to the conjugate-gradient method. Additionally, TNC allows for the inclusion of upper and lower bounds on the variables, making it suitable for solving constrained minimization problems. In the hybrid QML algorithm, where n is the size of the input, $f_i(n)$ is the time complexity function of the i-th sub-algorithm, and n_i is the number of times the i-th sub-algorithm is run, TNC can be used as a sub-algorithm to efficiently solve optimization problems.

2.3.10 Impact on Performance

Suryotrisongko and Musashi (2022) investigated hybrid quantum deep learning (DL) and VQC-based methods using IBM's superconducting qubit hardware. The authors simulated the performance of various combinations of key optimizers with variational forms and feature maps. The top performers are listed in Table 2.

Table 2: Maximum Accuracy of the Key Optimizer

Optimizer	Parameterized Circuit	FeatureMap	Accuracy (%)
CG	TwoLocal, RealAmplitudes, EfficientSU2	RawFeatureVector	84.4
COBYLA	TwoLocal, RealAmplitudes, EfficientSU2	RawFeatureVector	84.4
L_BFGS_B	TwoLocal, RealAmplitudes, EfficientSU2	RawFeatureVector	84.4
NELDER_MEAD	TwoLocal, RealAmplitudes	RawFeatureVector	84.4
TNC	RealAmplitudes, EfficientSU2	RawFeatureVector	84.4
P_BFGS	TwoLocal, RealAmplitudes, EfficientSU2	RawFeatureVector	84.4
POWELL	TwoLocal, RealAmplitudes, EfficientSU2	RawFeatureVector	84.4
SLSQP	TwoLocal, RealAmplitudes, EfficientSU2	RawFeatureVector	84.4
NFT	TwoLocal	ZFeatureMap	78.0
AQGD	EfficientSU2	ZFeatureMap	76.4

2.4 Domain Generation Algorithm Botnets, Quantum Machine Learning, and the Knowledge Gap

In their 2021 study, Suryotrisongko and Musashi explored QML for cybersecurity, specifically focusing on the detection of DGAs. The study examined two different quantum ML methods: a hybrid approach that combines quantum and classical DL, and the VQC model. The research involved comprehensive benchmarking of these methods, using quantum noise models, and comparing their respective performances. The experimental setup used extensive datasets that included statistical analyses such as entropy, RE, information radius, and reputation scores. The datasets were derived from a large collection of 1,000,000 Alexa domain names and 803,333 DGA domain names. To conduct the quantum simulations, the researchers utilized prominent quantum computing frameworks, namely Qiskit and PennyLane (Suryotrisongko & Musashi, 2021).

In a subsequent paper, Suryotrisongko et al. (2022) reported that the hybrid quantum DL model outperformed the VQC model, achieving an impressive accuracy rate of 94.9%. The study involved a meticulous analysis incorporating twelve distinct optimizer algorithms, four feature maps, and four variational form circuits. Notably, their research shed light on the pivotal role of feature maps and variational forms in determining the efficacy of the VQC algorithm. Furthermore, the authors highlighted the effectiveness of a quantum circuit architecture that combines angle embedding and random layers within the hybrid quantum DL model (Suryotrisongko et al., 2022).

However, no research has been conducted to address DGA Botnet detection using QSVM, PegasosQSVC, QNNs, or even the VQC on real quantum computing devices. This creates a significant knowledge gap in the existing literature. Additionally, no previous research has explored the application of these methods on non-IBM quantum architecture.

Chapter 3 - Methodology

3.1 Introduction

The methodology used in this praxis was designed to evaluate the potential benefits of HQBCs compared to established CBCs. The process consists of six distinct steps, each with a specific purpose in the overall research agenda.

1. Benchmark selection – The first step involves selecting a benchmark to evaluate the performance of HQBCs. CBCs are used as this benchmark, and their performance metrics are assessed using a unique sample from the dataset under consideration.
2. Simulator experiments – After benchmarking, the chosen HQBCs are executed on various quantum simulators, using the same data sample as CBCs. Due to the computational limitations of simulators, architectural hardening is necessary for the successful execution of these experiments.
3. Hardware experiments – The algorithms are then deployed on actual quantum hardware, requiring the implementation of architectural hardening once again to ensure successful execution. During this phase, it is important to incorporate mechanisms to handle potential interruptions, such as device failure, network issues, or device downtime, in order to maintain the integrity of the experimental process.
4. Hypothesis testing and evaluation – After collecting data, hypothesis testing is conducted to statistically evaluate the performance of HQBCs compared to CBCs. This step aims to determine whether the HQBCs offer significant advantages in terms of speed or accuracy.

5. Bottleneck analysis – A comprehensive analysis is performed to identify the limitations and requirements for the successful implementation of quantum algorithms in binary classification tasks. This analysis provides insights into the computational and architectural challenges that need to be addressed.
6. Algorithmic proposal and evaluation – Finally, a novel algorithm called QHTC is proposed to overcome the identified bottlenecks and limitations. The algorithm undergoes rigorous testing to evaluate its performance, and the null hypothesis is subsequently examined in the context of this new approach.

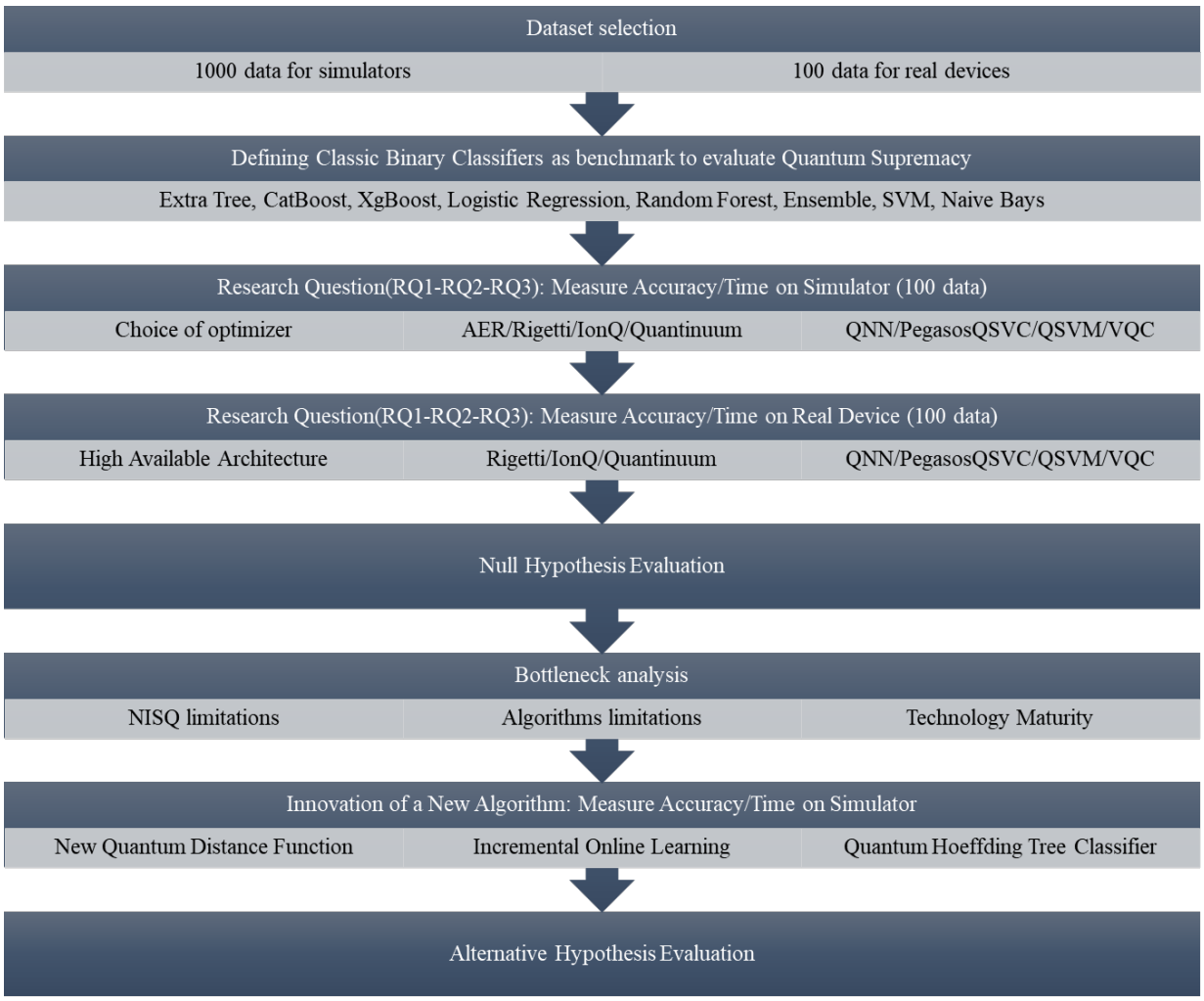

Figure 6: Research Methodology

In Figure 6, R_i (i = 1,2,3) represents the HQBCs, with R_1 addressing QSVC and PegasosQSVC, R_2 representing QNN, and R_3 evaluating the VQC. This systematic approach aims to contribute to a comprehensive understanding of the current state and potential future directions for QML, particularly in the field of binary classification and its applications in DGA-based botnet detection.

3.2 Prerequisite Dataset

The primary dataset analyzed for the DGA botnet study included the "IEEE Botnet DGA Dataset" along with the researchers' source code (Suryotrisongko, 2020). This dataset played a pivotal role in presenting examples of VQC and Hybrid Quantum DL implemented using Qiskit and PennyLane, facilitating an assessment of the quantum advantage of HQBCs. To ensure a consistent comparison with the findings of Suryotrisongko and Musashi (2021), the same dataset was utilized for analysis, and the algorithms' performance was evaluated, as detailed in Section 4.2. The dataset preparation involved selecting the top 1 million domains from Alexa and 10 DGA-based botnets (Conficker, Cryptolocker, Goz, Matsnu, New_Goz, Pushdo, Ramdo, Rovnix, Tinba, and Zeus).

The attributes used for dataset analysis are as follows:

1. CharLength: This attribute represents the length of the domain name in terms of the number of characters.
2. Entropy Value: This metric quantifies the domain name's entropy, computed through Shannon's function based on the probability distribution of the characters in the domain name:

$$H(X) = -\sum_{i \in X} p_i log_2\ (p_i) \qquad \text{Equation 33}$$

3. RE: This feature measures the similarity or divergence of the character probability distribution of a domain name, in comparison to either DGA domain names or Alexa's listings. It is calculated using the Kullback–Leibler divergence function (Kullback & Leibler, 1951). This function calculates the divergence, with "P" representing the character distribution in the target domain name and "Q" representing the probability distributions it is compared to:

$$D_{KL}(P \parallel Q) = \sum_{i=1}^{n} P(x_i) log_2 \frac{P(x_i)}{Q(x_i)} \qquad \text{Equation 34}$$

4. MinREBotnet: This attribute calculates the minimum RE in comparison with the domain names of various DGA botnets.
5. IRad: The IRad metric is calculated using the Jensen–Shannon divergence function (Nielsen, 2019). It evaluates the similarity or distance of a domain name's probability distribution in relation to multiple other distributions, specifically those of the 10 DGA botnet families' domains:

$$JSD_{\pi 1,\ldots,\pi_n}(P_1, \ldots, P_n) = H\left(\sum_{i=1}^{n} \pi_i P_i\right) - \sum_{i=1}^{n} \pi_i H(P_i) \qquad \text{Equation 35}$$

6. TreeNewFeature: This feature was generated by a decision tree algorithm, leveraging attributes such as Entropy, REAlexa, MinREBotnets, and CharLength.

These features were integrated within the decision tree to train a predictive model, resulting in the creation of TreeNewFeature.

7. ReputationAlexa: This feature indicates the popularity and credibility of the website in question (Zhao et al., 2019).

The dataset IEEE Botnet DGA consists of a total of 1,803,333 data records. To conduct the experimental analysis in this praxis, a random sample of 1,000 observations was chosen from this extensive dataset. The descriptive statistics for these features are presented in Table 3.

Table 3: Descriptive Statistics (Tehrani et al., 2023)

Variable	Mean	StDev	Min.	Median	Max.	Skewness	Kurtosis
CharLength	17.20	6.82	4.00	16.00	73.00	0.81	0.02
Entropy	3.02	0.53	0.00	3.04	4.78	-0.40	0.83
RelativeEntropy	1.66	0.82	0.20	1.55	10.10	1.63	6.91
MinREBotnets	1.28	0.57	0.00	1.23	5.99	0.84	1.24
InformationRadius	0.65	0.11	0.24	0.65	1.17	0.34	0.12
TreeNewFeature	0.45	0.34	0.00	0.35	0.99	0.38	-1.52
Reputation	81.66	54.12	0.00	64.51	436.31	0.99	0.21

A dataset known as UMUDGA (Zago et al., 2020) was also considered for this praxis. This dataset consists of 50 malware samples and is well-suited for multiple classifications using VQC. Although it was not used in the current experiments, it could be considered for future research steps.

3.3 Classic Binary Classifiers as Benchmarks

This praxis has involved the code used to conduct ten-fold cross-validation experiments for classical binary classifiers while maintaining a consistent sample size of 1,000 observations. The results of these experiments are presented in Table 4:

Table 4: Accuracy and Standard Deviation of Different Classic Binary Classifiers

Classical Binary Classifier	Accuracy	Standard Deviation
Extra Tree	0.91	0.05
CatBoost	0.91	0.04
XgBoost	0.91	0.06
Logistic Regression	0.906667	0.05
Random Forest	0.9	0.05
Ensemble	0.893333	0.07
SVM	0.876667	0.07
Naive Bayes	0.836667	0.07

Since the algorithms are already familiar, detailed mathematical explanations are omitted here. However, the modifications to the SVM, which serves as the foundation for hybrid quantum methods, will be explained in the next section.

Table 5: Accessible Devices

Device Name	Quantum Computer ID	# of Qubits	Supplier
H1-1 Syntax Checker	quantinuum.sim.h1-1sc	20	Quantinuum
H1-2 Syntax Checker	quantinuum.sim.h1-2sc	20	
H2-1 Syntax Checker	quantinuum.sim.h2-1sc	32	
H1-1 Emulator	quantinuum.sim.h1-1e	20	
H1-2 Emulator	quantinuum.sim.h1-2e	20	
H2-1 Emulator	quantinuum.sim.h2-1e	32	
H1-1	quantinuum.qpu.h1-1	20	
H1-2	quantinuum.qpu.h1-2	20	
H2-1	quantinuum.qpu.h2-1	32	
Rigetti QVM (Simulator)	rigetti.sim.qvm	20	Rigetti
Aspen M-3 (Hardware)	rigetti.qpu.aspen-m-3	80	
Quantum Simulator	ionq.simulator	29	IonQ
Aria 1	ionq.qpu.aria-1	23	
Quantum Computer	ionq.qpu	11	

3.4 Simulators, Real Devices, and the Role of Optimizers

In this praxis, a diverse range of quantum devices provided by Azure Quantum was utilized. Typically, within the general computational framework for quantum algorithms, classical bit representation data is transformed into a quantum bit (qubit) format. This transformation includes data encoding. Subsequently, the encoded data undergoes a specific type of manipulation, often referred to as “projection,” based on the algorithm being used. A loss value is then calculated by comparing this projection with the actual data. The objective is to minimize this loss, which is achieved through the use of optimization algorithms, commonly known as optimizers.

As part of the methodology for this praxis, a comprehensive examination of various optimizers was conducted to determine if there are significant mathematical differences between them. In the following sections, a summarized review of these optimizers is presented.

3.5 Selection of Hybrid Quantum Binary Classifiers

3.5.1 Deployment and Hardening for Training Stage

The training stage consists of a variety of classical algorithms and hybrid QML algorithms that are combined in different ways. Figure 7 illustrates the architecture of our training stage.

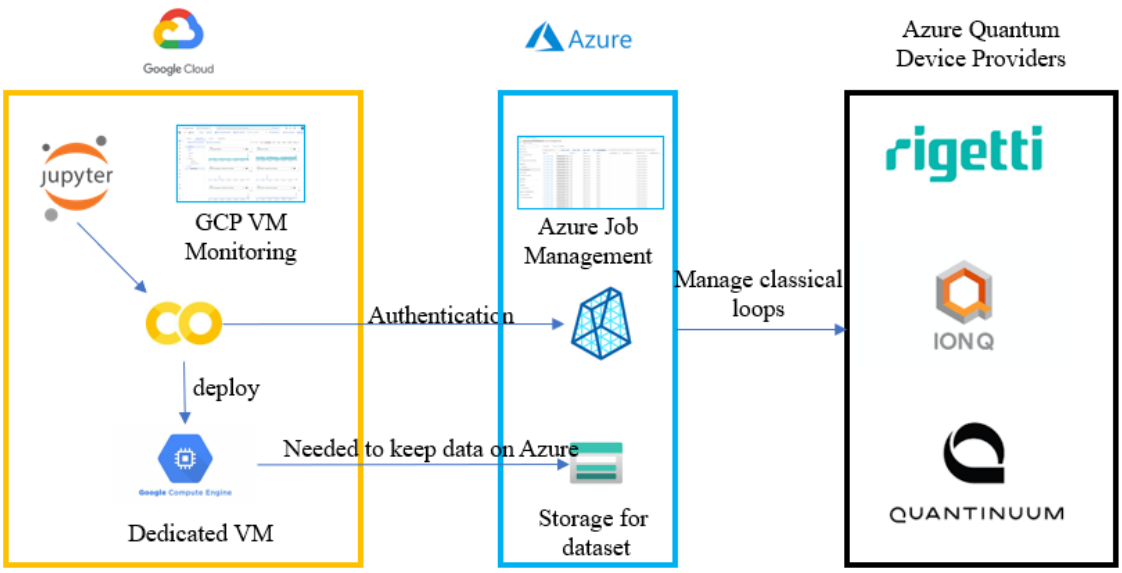

Figure 7: Stable Deployment Model after Bottleneck Analysis

3.5.2 Model Serving and Applicability of Model on Production

The Bring Your Own Model capability in Sentinel was used to deploy the model created during the training stage to the production stage. Subsequently, all relevant logs were transferred to the Sentinel Workspace. Figure 8 depicts this process in an Azure Kubernetes Service environment.

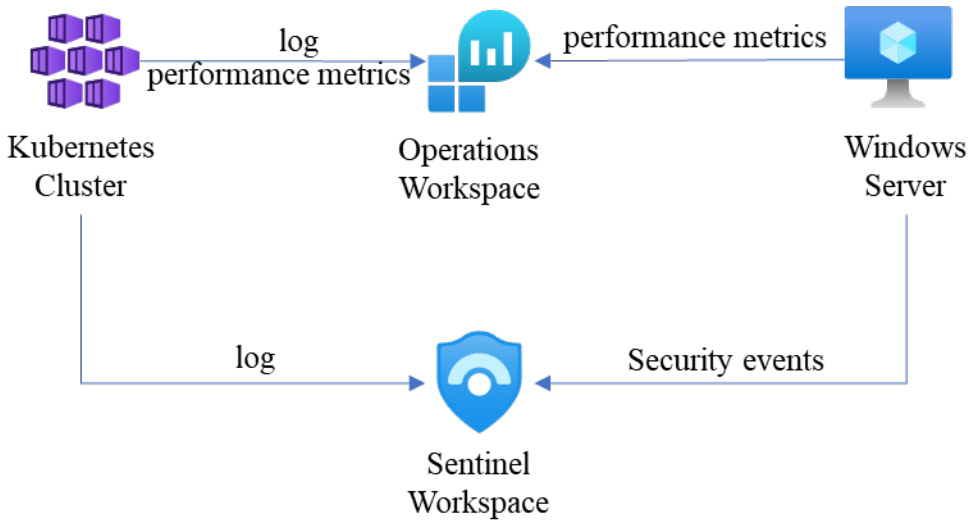

Figure 8: Model Serving

3.6 A New Algorithm: The Missing Step for NISQ Era

3.6.1 Distance Function on Multiqubit System

The QHTC-Distance calculation process is inspired by the scenario depicted in Figure 9, where all three qubits are imagined residing within a single Bloch sphere. Our function aims to articulate the concept of "distance" within these three quantum states in one hypothetical Bloch sphere.

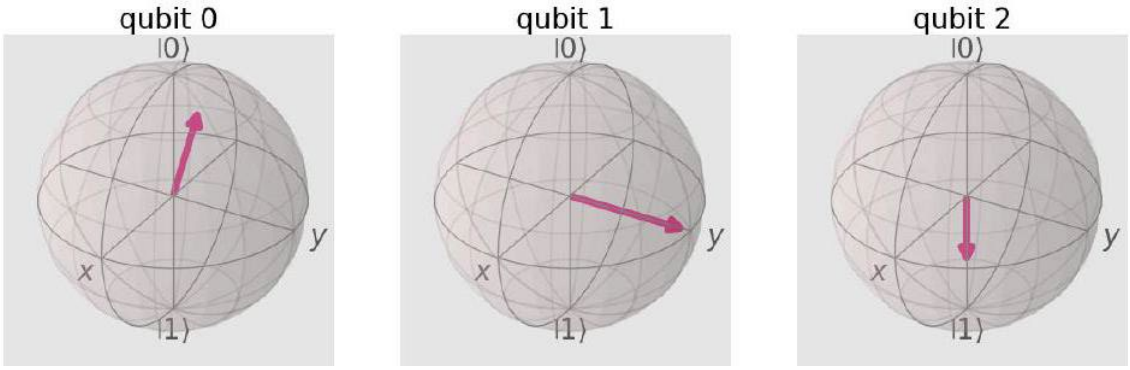

Figure 9: Imagining Three Qubits in a Bloch Sphere for New Distance Function

The QHTC-Distance calculation process involves five steps: feature mapping, wavefunction computation, density matrix calculation, trace distance calculation, and sum of distances.

First, the $z_$feature_map () function is responsible for encoding classical data into quantum states. Classical data points $x_1, x_2, \ldots, x_n$ are encoded using a series of gates (Hadamard and Z-rotation) on a quantum register of n qubits. The operation can be expressed as:

$$\begin{matrix} |\Psi_{\text{feature}}\rangle = \\ H^{\otimes n}\big(RZ(2\pi x_1) \otimes RZ(2\pi x_2) \otimes \ldots \otimes RZ(2\pi x_n)\big) \\ H^{\otimes n}|0\rangle^{\otimes n} \end{matrix} \qquad \text{Equation 36}$$

where $H^{\otimes n}$ denotes the Hadamard operation on all qubits, and $x_1, x_2, \ldots, x_n$ are the classical data points to be encoded.

3.6.2 Wavefunction Computation

For each quantum state $|\Psi_{\text{feature}}\rangle$, the relevant wavefunction ψ_{feature} is computed.

3.6.3 Density Matrix Calculation

Each quantum state $|\Psi_{\text{feature}}\rangle$ is then represented by its density matrix ρ_{feature}, computed as:

$$\rho_{\text{feature}} = |\Psi_{\text{feature}}\rangle\langle\Psi_{\text{feature}}| = \psi_{\text{feature}}\,\psi^{\dagger}_{\text{feature}} \qquad \text{Equation 37}$$

In the above equation, $|\Psi_{\text{feature}}\rangle\langle\Psi_{\text{feature}}|$ denotes the outer product of the state vector with its own conjugate transpose (bra vector), resulting in a matrix.

3.6.4 Trace Distance Calculation

The trace distance $D(\rho_1, \rho_2)$ between two quantum states (represented by their density matrices ρ_1 and ρ_2) is given by:

$$D(\rho_1, \rho_2) = \frac{1}{2}\sum_i |\lambda_i| \qquad \text{Equation 38}$$

where λ_i are the singular values of the matrix $\rho_1 - \rho_2$. The singular values are the square roots of the eigenvalues of the matrix $(\rho_1 - \rho_2)^\dagger(\rho_1 - \rho_2)$.

The final output produced by the QHTC_Distance_Rigetti() function is the cumulative sum of all trace distances calculated for each pair of consecutive quantum states, as well as the pair consisting of the initial and final quantum states:

$$\text{Sum of Distances } = \sum_{i=1}^{n-1} D(\rho_i, \rho_{i+1}) + D(\rho_n, \rho_1) \qquad \text{Equation 39}$$

3.6.5 PyQuil Example

The code snippet below demonstrates the implementation of the mentioned methods using Rigetti's PyQuil to turn the new metric in Equation 39 into code:

```
#make distance function
from pyquil import Program, get_qc
from pyquil.gates import RZ, H
from pyquil.api import WavefunctionSimulator
import numpy as np
from scipy.linalg import svdvals
def z_feature_map(data): # ZFeatureMap function
    program = Program()
    qubits = len(data)
    for q in range(qubits):
        program += H(q)
    for q in range(qubits):
        program += RZ(2 * np.pi * data[q], q)
    for q in range(qubits):
        program += H(q)
    return program
# Trace distance function
def trace_distance(dm1, dm2):
    diff = dm1 - dm2
    singular_values = svdvals(diff)
    return 0.5 * np.sum(np.abs(singular_values))
def QHTC_Distance_Rigetti(parameters):
    # Connect to the QVM
    qc = get_qc('7q-qvm')
    # Create a wavefunction simulator
    wfn_sim = WavefunctionSimulator()
    # Create a ZFeatureMap for each qubit with corresponding parameters
    programs = [z_feature_map(param) for param in parameters]
    # Get the wavefunctions for each program
    wavefunctions = [wfn_sim.wavefunction(program) for program in programs]
    # Compute the density matrices for each state
    density_matrices = [np.outer(wf.amplitudes, np.conj(wf.amplitudes)) for wf in
wavefunctions]
    # Compute the trace distance for each adjacent pair of qubits and the pair of the last
and the first qubit
    distances = []
    for i in range(6):
        trace_distance_value = trace_distance(density_matrices[i], density_matrices[i+1])
        distances.append(trace_distance_value)
    trace_distance_value = trace_distance(density_matrices[6], density_matrices[0])
    distances.append(trace_distance_value)
    return sum(distances)
```

3.6.6 Qiskit AER Example

The code snippet below provides an implementation of the discussed methods using Qiskit's AER:

```
#distance for AER
from qiskit import QuantumCircuit, transpile, Aer, execute
from qiskit.circuit.library import ZFeatureMap
from qiskit.visualization import plot_bloch_multivector
from qiskit.quantum_info import Statevector, partial_trace, DensityMatrix, state_fidelity
from scipy.linalg import svdvals
import numpy as np
parameters = [[1],[2],[3],[4],[5],[6],[7]]
# Trace distance function
def trace_distance(dm1, dm2):
    diff = dm1 - dm2
    singular_values = svdvals(diff.data)
    return 0.5 * np.sum(np.abs(singular_values))
def QHTC_Distance_AER(parameters):
    qc = QuantumCircuit(7)
    for i in range(7):
        zfm = ZFeatureMap(feature_dimension=1, reps=1)
        zfm_qc=zfm.bind_parameters(parameters[i])
        qc.append(zfm_qc, [i])
    qc = transpile(qc)
    sv_sim = Aer.get_backend('statevector_simulator')
    job_sim = execute(qc, sv_sim)
    statevec = job_sim.result().get_statevector()
    dms = []
    for i in range(7):
        reduced_dm = partial_trace(Statevector(statevec), [j for j in range(7) if j != i])
        dms.append(DensityMatrix(reduced_dm.data))
    distances = []
    for i in range(6):
        trace_distance_value = trace_distance(dms[i], dms[i+1])
        distances.append(trace_distance_value)
    trace_distance_value = trace_distance(dms[6], dms[0])
    distances.append(trace_distance_value)
    return sum(distances)
print(f"QHTC Distance: {QHTC_Distance_AER(parameters)}")
```

3.7 Quantum Hoeffding Tree Classifier

QHTC is an HTC that uses the Quantum distance function described above using data encoded as qubits with the ZFeatureMap. The original HTC is shown below in Figure 10. The algorithm can be modified to incorporate online incremental learning using the code in Figures 11 and 12. By combining the principles in the provided code with the proposed distance function, a new QHTC algorithm is illustrated in Figure 13.

Algorithm 1 The Original Hoeffding Tree Algorithm [35] (HTC)

Inputs:

- S is a sequence of examples.
- $\mathbf{X}$ is a set of discrete attributes.
- $G(\cdot)$ is a split evaluation function.
- δ is one minus the desired probability of choosing the correct attribute at any given node.

Output:

- HT is a decision tree.

procedure HoeffdingTree($S, \mathbf{X}, G, \delta$)
  Let HT be a tree with a single leaf l_1 (the root).
  Let $\mathbf{X}_1 = \mathbf{X} \cup \{X_\emptyset\}$.
  Let $\bar{G}_1(X_\emptyset)$ be the $\bar{G}$ obtained by predicting the most frequent class in S.
  for each class y_k **do**
    for each value x_{ij} of each attribute $X_i \in \mathbf{X}$ **do**
      Let $n_{ijk}(l_1) = 0$.
    end for
  end for
  for each example $(\mathbf{x}, y_k)$ in S **do**
    Sort $(\mathbf{x}, y)$ into a leaf l using HT.
    for each x_{ij} in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$ **do**
      Increment $n_{ijk}(l)$.
    end for
    Label l with the majority class among the examples seen so far at l.
    if the examples seen so far at l are not all of the same class **then**
      for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ **do**
        Compute $\bar{G}_l(X_i)$ using the counts $n_{ijk}(l)$.
      end for
      Let X_a be the attribute with the highest $\bar{G}_l$.
      Let X_b be the attribute with the second-highest $\bar{G}_l$.
      Compute $\epsilon = \sqrt{\frac{R^2}{2} \cdot \ln\left(\frac{1}{\delta}\right) \cdot \frac{1}{\sum_{ijk} n_{ijk}(l)}}$.
      if $\bar{G}_l(X_a) - \bar{G}_l(X_b) > \epsilon$ and $X_a \neq X_\emptyset$ **then**
        Replace l by an internal node that splits on X_a.
        for each branch of the split **do**
          Add a new leaf l_m, and let $\mathbf{X}_m = \mathbf{X} - \{X_a\}$.
          Let $\bar{G}_m(X_\emptyset)$ be the $\bar{G}$ obtained by predicting the most frequent class at l_m.
          for each class y_k and each value x_{ij} of each attribute $X_i \in \mathbf{X}_m - \{X_\emptyset\}$ **do**
            Let $n_{ijk}(l_m) = 0$.
          end for
        end for
      end if
    end if
  end for
  return HT.
end procedure

Figure 10: Implementation of Hoeffding Tree Classifier (Tehrani et al., 2023)

Algorithm 2 HoeffdingTreeClassifier (HTC) in its learning-mode version

procedure INIT($nFeatures$, $nClasses$, $delta = 0.01$, $tiethreshold = 0.05$)
    Store the input variables.
    $root \leftarrow$ TREENODE($delta$)
end procedure

function PREDICT(X)
    Predict the class labels for the input instances X.
    return $yPredict$
end function

procedure PARTIALFIT(X, y)
    Update the tree with new training instances X and their corresponding class labels y.
end procedure

procedure UPDATESTATISTICS($X, label$)
    Update the statistics of the tree nodes based on the input instance X and its class label $label$.
end procedure

procedure ATTEMPTSPLIT($node$)
    Attempt to split the given node $node$ based on the Hoeffding bound gain.
end procedure

procedure SPLITNODE($node$)
    Split the given node $node$ by selecting the best attribute based on the Hoeffding bound gain.
end procedure

function HOEFFDINGBOUND($node$)
    Split the given node $node$ by selecting the best attribute based on the Hoeffding bound gain.
    return $epsilon$
end function

Figure 11: Instrumentation of Hoeffding Tree Classifier for Online Learning

(Tehrani et al., 2023)

Algorithm 3 TreeNode (as part to algorithm 2)

procedure INIT(*delta*)
    Store the input variable.
    Initialize further variables.
end procedure

function ISLEAF
    Check if the node is a leaf (no children).
    return not *children*
end function

procedure COMPUTEERRORRATE
    Compute the error rate of the node based on the class distribution.
end procedure

procedure COMPUTEBESTSPLITTINGATTRIBUTE(*nClasses*, *nFeatures*)
    Compute the best attribute to split on based on the Hoeffding bound gain.
end procedure

function COMPUTEHOEFFDINGBOUNDGAIN(*classCountsPerValues*, *totalSamplesPerValue*, *nClasses*, *totalSamples*)
    Compute the Hoeffding bound gain for the given attribute values and their class counts.
    return *gain*
end function

function HOEFFDINGBOUND(*errorRate*)
    Compute the Hoeffding bound for the given error rate *errorRate*.
    return $hoeffdingBound$
end function

function GETMAJORITYCLASS
    Get the majority class label based on the class distribution.
    return *majorityClass*
end function

Figure 12: Additional Code for Online Learning with Hoeffding Tree Classifier

(Tehrani et al., 2003)

Algorithm 4 Batch-wise Learning with the Quantum-enhanced Hoeffding Tree Classifier (QHTC)

procedure RunQHTC
    Load the data set.
    Extract the features and labels from the data set.
    Normalize the features using a standard scaler.
    Initialize a few variables.
    for each feature row r in data set **do**
        Compute quantum walk distance QHTCDistanceRigetti(r).
    end for
    Store the new quantum data rows as a combination of the quantum walk distances and labels.
    Initialize number of features $nFeatures$ with 1.
    Initialize number of classes $nClasses$ with number of unique values in labels.
    Initialize function HoeffdingTree.init($nFeatures$, $nClasses$).
    Set the percentage for quantum test data.
    Set the batch size.
    for all feature rows with step batch size **do**
        Split the quantum data into training and testing data.
        Fit the function HoeffdingTree.partialFit($XTrain, yTrain$) to the quantum training data.
        Predict labels for the quantum test data using the function HoeffdingTree.predict($XTest$).
        Calculate all performance metrics of the predicted labels $yPredict$.
    end for
    Plot performance metrics per batch.
end procedure

function QHTCDistanceRigetti(r)
    Connect to Rigetti-S using library pyquil or to Aer using library qiskit.
    Create a wave function simulator ws.
    for each data point p in feature row r **do**
        Compute qubit q as ZFeatureMap transformation of p using library pyquil (Rigetti-S) or qiskit (Aer).
    end for
    for each qubit q **do**
        Get the wave functions $f(q)$ from ws.
    end for
    for each wave function $f(q)$ **do**
        Compute the density matrices $m_{f(q)}$.
    end for
    for $i \leftarrow 1$ to n while $i < n$ and n is length of feature row r **do**
        Let single distances d_i = TraceDistance($m_{f(q)}(i), m_{f(q)}(i+1)$).
    end for
    Let single distance d_n = TraceDistance($m_{f(q)}(n), m_{f(q)}(1)$).
    return Sum of single distances
end function

function TraceDistance(m_1, m_2)
    Let $d = m_1 - m_2$.
    Compute singular values s_i of d.
    return $0.5 * \sum_i |s_i|$
end function

Figure 13: Novel Hybrid Quantum Binary Classifier Algorithm (Tehrani et al., 2023)

3.8 Summary

The methodology used in this praxis is shown in the Figure 6. It is designed to provide a structured approach to answering the research questions. Importantly, this is the first time a methodology has been developed to analyze the fit between technology and problems in the field of QML and cybersecurity analytics.

By following this systematic framework, the praxis aims to gain a deep understanding of the current situation and future possibilities for QML, with a focus on binary classification tasks. The methodology consists of several stages, including the selection of benchmarks, setting up experiments on simulators and actual hardware, conducting hypothesis testing, analyzing bottlenecks, and proposing and evaluating a new algorithm.

Chapter 4 - Results

This section presents the results of the evaluations for each null hypothesis across real devices, simulators, and findings.

4.1 Hypothesis Evaluation for Real Quantum Devices

Table 6 displays the results that address the H0 corresponding to the first three research questions over actual quantum devices.

Table 6: Actual Device Time and Accuracy (Tehrani et al., 2023)

Algorithm	Accuracy (%)	Time(sec)	Optimizer	Device Provider
VQC	50.00	1,325,133	COBYLA	IonQ
PegasosQSVC	41.15	156,156	Pegasos	
QSVC	53.34	283,325	COBYLA	
QNN(Sampler)	56.00	956,540	COBYLA	
QNN-Estimator	59.00	1,165,819	COBYLA	
VQC	43.00	1,176,879	COBYLA	Rigetti
PegasosQSVC	48.02	355,509	Pegasos	
QSVC	39.37	385,153	COBYLA	
QNN-Sampler	52.75	1,601,895	COBYLA	
QNN-Estimator	51.44	1,437,085	COBYLA	
VQC	44.00	972,732	COBYLA	Quantinuum
PegasosQSVC	44.00	417,968	Pegasos	
QSVC	45.00	472,847	COBYLA	
QNN-Sampler	46.31	1,087,789	COBYLA	
QNN-Estimator	50.00	1,167,143	COBYLA	

In the empirical evaluation of the research questions, the null hypotheses were rejected when experiments were conducted using the actual device. Notably, the observed accuracy metrics were lower compared to those of existing CBCs.

Table 7: Classical Binary Classifiers as a Benchmark

Classic Binary Classifiers	Accuracy (%)	SD-Cross folder 10
Extra Tree	91.00	0.05
CatBoost	91.00	0.04
XgBoost	91.00	0.06
Logistic Regression	90.66	0.05
Random Forest	90.00	0.05
Ensemble	89.33	0.07
SVM	87.66	0.07
Naive Bayes	83.66	0.07

This observation is further supported by a comparative analysis of the descriptive statistics related to the accuracy of Hybrid Quantum Binary Classifiers (HQBCs) and CBCs. The data suggest that the performance of HQBCs is not yet on par with their classical counterparts:

Table 8: Comparison of Hybrid Quantum Binary Classifiers Versus Classical Binary Classifiers

Accuracy	Real Device Hybrid Quantum Binary Classifiers (%)	Classic Binary Classifiers (%)
Mean	48.23	89.29
Standard Error	1.46	0.90
Median	48.02	90.33
Mode	50.00	91.00
Standard Deviation	5.64	2.55
Sample Variance	31.81	6.50
Kurtosis	–0.67	3.54
Skewness	0.28	-1.89
Range	19.63	7.34
Minimum	39.37	83.66
Maximum	59.00	91.00
Sum	723.38	714.31
Count	15.00	8.00

First, these devices often experience downtime, which requires limiting the size of the dataset to ensure calculations can be done within a short timeframe. As a result, it is necessary to save intermediate results to facilitate the resumption of computations after downtime. This issue was the motivation behind our proposal for an incremental learning algorithm designed to adapt to these limitations. Second, the lack of built-in exception-handling features in the Qiskit library for non-IBM devices means that custom exception-handling mechanisms must be incorporated into the codebase. This is essential to handle unexpected failures that may occur during lengthy computational runs.

Third, maintaining an open authentication session over extended periods presents an additional challenge for long-duration calculations. This obstacle can be overcome through careful programming practices, such as session re-authentication or token refreshing.

Additionally, these practical limitations can sometimes result in data loss, which can lead to lower accuracy rates in the final results. In addition to these operational challenges, the quality of quantum gates and qubits themselves is still an open area of research.

Given these constraints and limitations, it is unrealistic to expect significant improvements in accuracy from existing algorithms. This highlights the need for the development of novel algorithms that take these real-world challenges into account. The development of such algorithms could lead to more robust and efficient implementations of QML, making meaningful advancements in the field.

The perception that quantum computers are inherently faster for all computational tasks is a complex issue that requires careful consideration. While it is true that quantum algorithms, such as Shor's algorithm, have shown potential to outperform classical algorithms in specific areas—particularly in the factorization of large integers, which has

significant implications for breaking Rivest–Shamir–Adleman (RSA) cryptography. In Shor's algorithm, the quantum Fourier transform is used in the context of phase estimation to efficiently calculate the period of a discrete logarithm function. With high-quality qubits, it is projected that Shor's algorithm could factorize RSA 2048-bit keys in just 20 minutes, whereas classical computers would take millions of years using brute-force methods.

However, the speed advantages of quantum computing do not apply universally to all computational problems, including QML tasks. In the case of the HQBCs examined in this praxis, the quantum circuits used did not utilize quantum phenomena like superposition to gain a computational advantage. As a result, these HQBCs exhibited significant inefficiencies: while CBCs could complete the task in seconds, the average time required to train a model with only 100 data elements using HQBCs was approximately 864,131.53 seconds. Such computational costs are impractically high, making this approach unsuitable for real-world applications, especially in the field of cyber defense.

4.2 Hypothesis Evaluation Utilizing Quantum Simulators

Table 9 presents the results addressing the mentioned H0 for the research questions regarding simulators of quantum devices. Two deployment options are available for simulators. Some, like AER (similar to a fault-tolerant quantum computer on a simulation scale) or Rigetti (NISQ computer on a simulation scale), can be hosted on a container on a normal laptop, ensuring consistent and fast calculation results. However, many simulators can only be accessed through cloud service providers. In our case, to ensure a fair comparison, simulators were deployed and made available on the Azure Cloud.

Table 9: Results of Hybrid Quantum Binary Classifiers Over Simulators

Algorithm	Accuracy	Total (sec)	Optimizer	FeatureMap	Simulators
VQC	54	4240	COBYLA	ZFeatureMap	AER
PegasosQSVC	90	45	Pegasos	ZFeatureMap	
QSVC	87	3091	COBYLA	ZFeatureMap	
QNN(Sampler)	76	374	COBYLA	ZFeatureMap	
QNN(Estimator)	84	410	COBYLA	ZFeatureMap	
VQC	51	957755	COBYLA	ZFeatureMap	IonQ
PegasosQSVC	49	113950	Pegasos	ZFeatureMap	
QSVC	57	178529	COBYLA	ZFeatureMap	
QNN(Sampler)	59	746992	COBYLA	ZFeatureMap	
QNN(Estimator)	63	780480	COBYLA	ZFeatureMap	
VQC	46	889708	COBYLA	ZFeatureMap	Rigetti
PegasosQSVC	55	206729	Pegasos	ZFeatureMap	
QSVC	45	205877	COBYLA	ZFeatureMap	
QNN(Sampler)	58	656629	COBYLA	ZFeatureMap	
QNN(Estimator)	54	955654	COBYLA	ZFeatureMap	
VQC	45	806626	COBYLA	ZFeatureMap	Quantinuum
PegasosQSVC	49	174416	Pegasos	ZFeatureMap	
QSVC	49	197871	COBYLA	ZFeatureMap	
QNN(Sampler)	48	852774	COBYLA	ZFeatureMap	
QNN(Estimator)	53	716581	COBYLA	ZFeatureMap	

Considering the experimental results obtained from quantum simulators, it is evident that the null hypothesis concerning the performance of Hybrid Quantum Binary Classifiers (HQBCs) could be rejected. This reinforces the notion that the current state-of-the-art in QML algorithms, at least in the context of this praxis, has yet to match or surpass the performance benchmarks set by CBCs.

Furthermore, the accuracy metrics observed for the HQBCs were found to be inferior to those of the CBCs, adding another layer of complexity to the assessment of their practical utility. One might argue that a quantum approach could offer advantages in terms of computational speed or tolerance to larger data samples; however, the experimental data

do not support this assertion. Despite demonstrating relatively better tolerance for larger datasets—up to 1000 data samples—the HQBCs did not exhibit any speed advantages over their classical counterparts.

These findings present a compelling case for re-evaluating the efficacy and applicability of existing QML algorithms, mainly HQBCs, particularly in comparison to established classical methods.

4.3 Bottleneck Analysis

Quantum computers have well-known limitations, such as low-quality qubits and gates, the absence of error correction and quantum memory, and challenges in managing quantum data in transit. While these limitations are known, this praxis reveals the practical execution challenges implicit in NISQ era quantum computing hardware:

- Challenges with real quantum devices:
    - A single-circuit run failure can have a cascading effect, regardless of any progress made.
    - Bugs in task queue prioritization and scheduling.
    - Periods of maintenance downtime.
    - Difficulties in implementing quantum cloud architecture on a small scale due to limited or outdated documentation.
- Difficulties with Azure Quantum Workspace's hosted Jupyter Notebooks:
    - Kernel crashes.
    - Inadequate memory capacity.
    - Limited number of virtual CPUs.
    - Poor visibility of progress and log processing.

- Communication hurdles between real quantum devices and Notebooks:
    - Authentication and session disruptions.
- Client-side Jupyter Notebook issues:
    - Automatic termination after 24 hours, regardless of CPU or RAM capacity.
- Issues specific to various real devices:
    - Outdated APIs in Qiskit.
- Problems arising from the algorithm's nature:
- Excessively numerous loops.
- Limited code portability.
- Subpar exception handling.

To address these deployment challenges, a strategic design approach was adopted, as illustrated in Figure 14, and advanced exception handling was incorporated into the code.

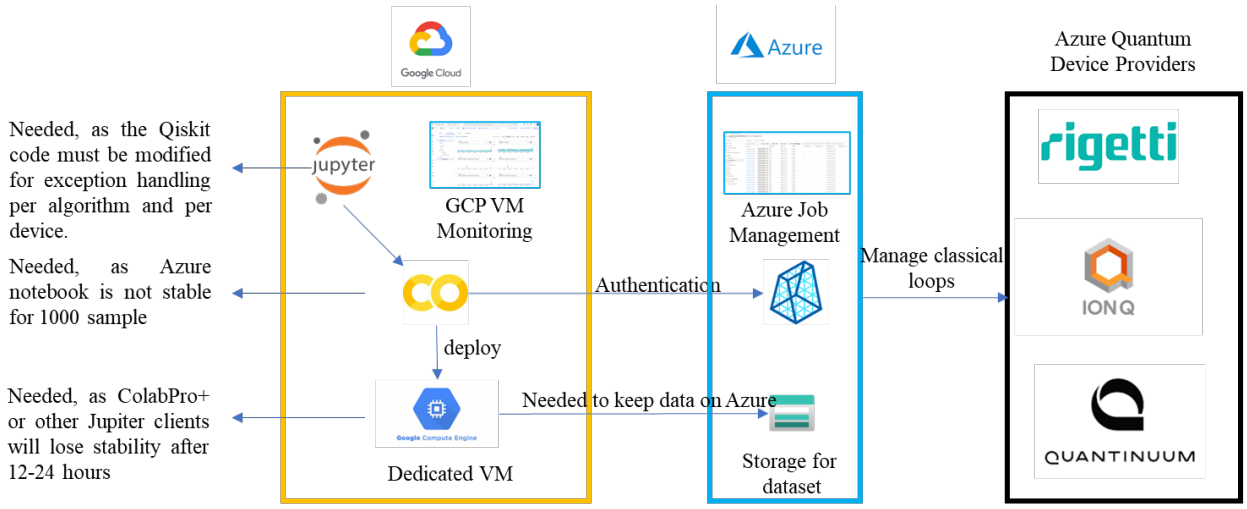

Figure 14: Deployment Hardening as Part of the Solution

4.4 New Quantum Hoeffding Tree Classifier Algorithm

The praxis presents the results of the QHTC algorithm, which was executed in five separate batches, each consisting of 1,000 randomly selected fixed data samples, using the ZFeatureMap feature map from Qiskit. As shown in Table 10, there was a significant improvement in accuracy, with the accuracy increasing as more batches were processed, as expected. Surprisingly, the QHTC algorithm performed consistently well across all three feature maps. It achieved an average accuracy rate of 91.2% and a perfect accuracy of 100% in the final round of QHTC, even though only 3 out of the planned 5 batches were completed.

To ensure a robust comparison, the same features and dataset as those used in the reference (Suryotrisongko & Musashi, 2022) were used. The high level of accuracy may be attributed to these features representing the entire dataset. In future research, the methodology will be further refined. This will involve improving the computation metrics to avoid overfitting and enhance the authenticity of the model. Techniques such as principal component analysis and k-fold cross-validation (with k set to 10) will be employed for each batch. Furthermore, it has been observed that the features EntropyValue and RelativeEntropy have significant predictive capabilities across the entire dataset. Therefore, achieving such a high level of accuracy may not be possible with other datasets that do not possess similar predictive characteristics.

Table 10: Quantum Hoeffding Tree Classifier Test Results (Tehrani et al., 2003)

Batch	Accuracy (%)	F1-score (%)	AUC (%)
1	57.1	4.5	51.1
2	99.0	98.8	98.9
3	100.0	100.0	100.0
4	100.0	100.0	100.0
5	100.0	100.0	100.0
Average	91.2	80.7	90.0

Despite the limitations that HQBCs present regarding instrumentalization, there is at least one scenario in which a QHBC outperforms the current CBCs. Therefore, the alternative hypothesis is proven to be valid.

4.5 Evaluation of Results

Selection of metrics: The selection of metrics for evaluation involved defining accuracy as the percentage of correct predictions, while speed was measured in terms of the computational time required to generate a model. Given that current quantum devices are exclusively available through the cloud, the measurement of time is further divided into two categories: device time and cloud time. Device time is calculated using the formula:

$$QC_{Time} = \sum_{i=\ circuit\ calls} Circuit_Time_i \qquad \text{Equation 40}$$

Note: Cloud time is the duration of the task minus QC_{Time}

To ensure generalizability, a sample of 1,000 rows for simulators and 100 observations for the dataset for a real quantum device was chosen. These sample sizes strike a balance

between general applicability and manageability in terms of costs and time associated with experiments over real quantum machines. The dataset was further split into a 70% training set and a 30% test set. This division allows for training classifiers on one set of data and testing them on another to assess their generalization to unseen data.

Model training: For model training, both HQBCs and CBCs were trained on the dataset, utilizing specific classifiers as listed in Table 11.

Table 11: List of Classifiers Used

Classic Binary Classifiers	Hybrid Quantum Binary Classifiers
Extra Tree	PegasosQSVC
CatBoost	QSVC
XgBoost	VQC
Logistic Regression	SamplerQNN
Random Forest	EstimatorQNN
Ensemble	QHTC
SVM	
Naive Bayes	

The trained models were employed to predict the labels of the test set, and the time taken for each model to make these predictions was recorded. Accuracy and speed metrics were then calculated for each model based on QC-Time and Cloud-Time. A statistical test was conducted to compare the performance of the two types of classifiers. If the p-value from the statistical test was below a certain threshold (e.g., 0.05), the null hypothesis would be rejected, indicating a significant difference in accuracy and/or speed between HQBCs and CBCs. Conversely, if the p-value is above this threshold, the null hypothesis would not be rejected, suggesting no significant difference.

Based on the results, the null hypothesis—Existing HQBCs can be readily used with NISQ hardware to achieve comparable or superior results to CBCs—was rejected. To

address the research questions and test the alternate hypothesis, the QHTC (Tehrani et al., 2023) was developed and successfully executed.

Chapter 5 - Discussion and Conclusions

According to the definition provided by Schuld and Killoran (2022), supervised learning is intended to formulate a problem and its solution by employing a suitable input domain $\mathcal{X}$ and a label domain $\mathcal{Y}$ (Schuld & Killoran, 2022). It involves a probability distribution $p(x)$ over inputs $x \in \mathcal{X}$. The process assumes the existence of a ground-truth mapping $f^* : \mathcal{X} \to \mathcal{Y}$, which accurately maps inputs to their corresponding target labels. In this setting, there is a finite set of input samples drawn from $\mathrm{p(x)}$, each paired with their target labels $\mathcal{D} = \{(\mathrm{x}^1, \mathrm{y}^1), \ldots, (\mathrm{x}^\mathrm{M}, \mathrm{y}^\mathrm{M})\}$ where $(\mathrm{x}, \mathrm{y}) \in \mathcal{X} \otimes \mathcal{Y}$. Additionally, a loss function $\mathrm{l}: \mathcal{Y} \otimes \mathcal{Y} \to \mathbb{R}$ is defined to assess the accuracy of a function $\mathrm{f}: \mathcal{X} \to \mathcal{Y}$ in predicting labels compared to the target labels.

As proposed by Schuld and Killoran (2022), our objective is to find a model f from a class of model functions $\mathcal{F}$ that minimizes the expected loss over the data distribution,

$$\hat{\mathrm{f}} = \min_{\mathrm{f} \in \mathcal{F}} \int_{\mathcal{X}} \mathrm{p(x)l(f(x), f^*(x))dx} \qquad \text{Equation 41}$$

With empirical data, the above definition becomes the following for sample D such that:

$$\hat{\mathrm{f}}_{\mathrm{emp}} = \min_{\mathrm{f} \in \mathcal{F}} \frac{1}{|\mathcal{D}|} \sum_{(\mathrm{x},\mathrm{y}) \in \mathcal{D}} \mathrm{l(f(x), y)}. \qquad \text{Equation 42}$$

Therefore, QSVC, PegasosQSVC, QSVM, and VQC are quantum versions of such a strategy. There are three main approaches:

- Approach 1: Focus on minimizing the loss function by researching and evaluating relevant families of optimizers. Examples of this approach include QSVM (Rebentrost et al., 2014) and quantum DL (Zlokapa et al., 2021).

- Approach 2: Focus on f(x) and a model class of F with hypothesis (f(x) = y) such that QC provides the f(x). Hierarchical classifier (Zlokapa et al., 2021) and QNN (Farhi & Neven, 2018) are examples of this approach.
- Approach 3: Focus on the loss function in such a way that QC provides the parameters needed to calculate the loss. VQC (Havlíček et al., 2019) and PegasosQSVC (Qiskit 0.7.0, 2023a) can be used with this approach.

These methods involve transmitting x to a quantum circuit using feature maps and the possible calculation of parameters θ by another quantum circuit (like ansatz in VQC). NISQ devices have several shortcomings that must be considered regarding hybrid quantum machine learning (HQML):

1. Lack of high-quality gates and qubits.
2. Large CMOS.
3. Absence of quantum memory.
4. Absence of effective error correction methods.
5. Quantum data in transit is not feasible.
6. Limited qubits (logical ones are not the same as physical ones).
7. Circuits with high depth are slow and prone to errors.

Algorithms that require a large number of circuit calls will not provide a quantum advantage. This approach will likely fail due to accumulated errors from extended training times (Bittel & Kliesch, 2021).

QHTC circumvents an optimization problem by utilizing the derivative of a loss function. It focuses on the separability of data in a higher-dimensional space, constructs a classification tree, and strives to establish a new hypothetical distance that is related to

data in quantum space. This approach introduces a new path that needs to be explored in order to make quantum cybersecurity analytics more feasible for both NISQ and FTQC based computing platforms.

5.1 Future Research

Cybersecurity analytics plays a vital role in gathering data to create timelines, analyze threats, and develop proactive strategies against cyber threats. The rise of next-generation quantum cybersecurity analytics has led to a growing utilization of HQML in monitoring network activity, detecting changes in resource usage or network traffic, and responding effectively to threats. This advancement signifies the introduction of quantum-enhanced security information and event management systems (qSIEM). Our research showcases the first application of qSIEM in safeguarding against DGA-based botnet attacks that focus on the application layer in DDoS scenarios.

Despite these advancements, the current NISQ era presents challenges for qSIEM in cybersecurity, primarily due to inherent instabilities that intensify during prolonged computations. Our research proposes a novel form of HQML binary classifiers, resulting in improvements in accuracy and execution times in real-device–based simulations. This breakthrough involves the use of a quantum-enhanced version of the incremental Hoeffding tree algorithm, optimized for handling large volumes of online stream data and requiring fewer shots on real quantum devices.

This praxis successfully implemented HQML algorithms using 100 randomly selected fixed data samples over several weeks on Azure Quantum providers, using Qiskit. This combination is unprecedented in the field. However, future research should focus on improving the stability of quantum computations on actual devices beyond the small

portion of the IEEE Botnet DGA Dataset used in this praxis. Additionally, further investigation into quantum simulations is necessary, this praxis only utilized a sample size of 1,000 in HQBC and different sizes in QHTC for simulations on real devices.

Future research paths include:

- QHTC real device path:
    - Run QHTC on real devices.
- Quantum path:
    - Investigate the proposed distance function for quantum data.
    - Develop a resource roadmap for quantum cybersecurity analytics in NISQ and fault-tolerant quantum computing (FTQC) contexts.
- Algorithm path:
    - Explore hybrid quantum generative adversarial networks.
    - Delve into hybrid quantum unsupervised learning.
    - Advance hybrid quantum few-shot learning.
- Quantum cybersecurity analytics path:
    - Conduct more case studies on malware and spyware detection.
- Engineering path:
    - Enhance model serving with strong inference endpoints for qSIEM and evaluate performance in real-time.

Furthermore, the practical implementation of quantum-enhanced Hoeffding tree algorithms in other scientific domains is still unknown. The UMUDGA dataset is a potential subject for future research on DGA botnet detection.

Furthermore, this praxis suggests a more comprehensive exploration of HQBCs. The combination of quantum supervised learning and reinforcement learning, along with reward mechanisms, has the potential to generate groundbreaking cybersecurity tools. Even in the present NISQ era, which can support up to 5,600 qubits, there is the possibility of developing strong cybersecurity applications for current quantum computers and HQML. This is particularly important given that the number of qubits needed for network cybersecurity functions is lower than the available 5,600 qubits.

References

Anderson, R., Barton, C., Boehme, R., Clayton, R., Ganan, C., Grasso, T., Levi, M., Moore, T., & Vasek, M. (2019). Measuring the Changing Cost of Cybercrime. The 18th Annual Workshop on the Economics of Information Security, https://doi.org/10.17863/CAM.41598

Bittel, L., & Kliesch, M. (2021). Training variational quantum algorithms is NP-hard. Physical Review Letters, 127(12). https://doi.org/10.1103/PhysRevLett.127.120502

Brent, R. P. (2013). Algorithms for Minimization Without Derivatives (Dover Books on Mathematics) (English Edition) Kindle Edition.

BYRD, R. H., PEIHUANG LU, NOCEDAL, J., & CIYOU ZHU. (1995). A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific Computing, 16(5), 1190–1208. https://doi.org/10.1137/0916069

Chen, Z., & Subramanian, D. (2018). An Unsupervised Approach to Detect Spam Campaigns that Use Botnets on Twitter. arXiv: https://doi.org/10.48550/arxiv.1804.05232

Farhi, E., & Neven, H. (2018). Classification with Quantum Neural Networks on Near Term Processors. arXiv: https://doi.org/10.48550/arxiv.1802.06002

GRIPPO, L., LAMPARIELLO, F., & LUCIDI, S. (1989). A truncated Newton method with nonmonotone line search for unconstrained optimization. Journal of Optimization Theory and Applications, 60(3), 401–419. https://doi.org/10.1007/BF00940345

Havlíček, V., CÃ3rcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., & Gambetta, J. M. (2019). Supervised learning with quantum-enhanced feature spaces. Nature (London), 567(7747), 209–212. https://doi.org/10.1038/s41586-019-0980-2

Ianelli, N., & Hackworth, A. (2007). Botnets as a vehicle for online crime. The International Journal of Forensic Computer Science. https://doi.org/10.5769/j200701002

John, J. P., Moshchuk, A., Gribble, S. D., & Krishnamurthy, A. (2009, April). Studying spamming botnets using Botlab. Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009(pp. 291-306).

Karim, A., Salleh, R. B., Shiraz, M., Shah, S. A. A., Awan, I., & Anuar, N. B. (2014). Botnet detection techniques: review, future trends, and issues. Frontiers of Information Technology & Electronic Engineering, 15(11), 943–983. https://doi.org/10.1631/jzus.C1300242

Khan, W. Z., Khan, M. K., Bin Muhaya, F. T., Aalsalem, M. Y., & Han-Chieh Chao. (2015). A Comprehensive Study of Email Spam Botnet Detection. IEEE Communications Surveys and Tutorials, 17(4), 2271–2295. https://doi.org/10.1109/COMST.2015.2459015

Kraft, D. (1988). A software package for sequential quadratic programming. Forschungsbericht- Deutsche Forschungs- Und Versuchsanstalt Fur Luft- Und Raumfahrt.

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. The Annals of Mathematical Statistics, 22(1), 79–86. https://doi.org/10.1214/aoms/1177729694

Mitarai, K., Negoro, M., Kitagawa, M., & Fujii, K. (2018). Quantum circuit learning. Physical Review A, 98(3). https://doi.org/10.1103/PhysRevA.98.032309

Nazareth, J. L. (2009). Conjugate gradient method. Wiley Interdisciplinary Reviews. Computational Statistics, 1(3), 348–353. https://doi.org/10.1002/wics.13

Nielsen, F. (2019). On the Jensen–Shannon Symmetrization of Distances Relying on Abstract Means. Entropy (Basel, Switzerland), 21(5), 485-. https://doi.org/10.3390/e21050485

Nocedal, J. , & Wright, S. J. (2006). Numerical Optimization. Springer Nature. https://doi.org/10.1007/b98874

Olsson, D. M., & Nelson, L. S. (1975). The Nelder-Mead Simplex Procedure for Function Minimization. Technometrics, 17(1), 45–51. https://doi.org/10.1080/00401706.1975.10489269

Powell, M. J. (1994). A direct search optimization method that models the objective and constraint functions by linear interpolation (pp. 51-67). Springer Netherlands.

Putman, C. G. J., Abhishta, & Nieuwenhuis, L. J. M. (2018). Business Model of a Botnet. 2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 441–445. https://doi.org/10.1109/PDP2018.2018.00077

Qiskit 0.24.1. (2020a). Quantum-enhanced Support Vector Machine (QSVM) — Qiskit 0.24.1 documentation. Accessed June 15, 2023. https://qiskit.org/documentation/stable/0.24/tutorials/machine_learning/01_qsvm_classification.html

Qiskit 0.24.1. (2020b). Variational quantum classifier (VQC) — Qiskit 0.24.1 documentation. Accessed June 15, 2023. https://qiskit.org/documentation/stable/0.24/tutorials/machine_learning/03_vqc.html

Qiskit 0.7.0. (2023a). Pegasos quantum support vector classifier. Accessed June 15, 2023. https://qiskit.org/ecosystem/machine-learning/tutorials/07_pegasos_qsvc.html

Qiskit 0.7.0. (2023b). Qiskit: An open-source framework for quantum computing. Accessed June 15, 2023. https://doi.org/10.5281/zenodo.2573505

Qiskit 0.7.0. (2023c). Quantum neural networks – Qiskit machine learning 0.7.0. Accessed June 15, 2023. https://qiskit.org/ecosystem/machine-learning/tutorials/01_neural_networks.html

Rebentrost, P., Mohseni, M., & Lloyd, S. (2014). Quantum support vector machine for big data classification. Physical Review Letters, 113(3). https://doi.org/10.1103/PhysRevLett.113.130503

Schiller, C. A., Binkley, J., Harley, D., Evron, G., Bradley, T., Willems, C., & Cross, M. (2007). Botnets: The killer web applications. In Botnets: The Killer Web Applications. https://doi.org/10.1016/B978-1-59749-135-8.X5000-8

Schuld, M., & Killoran, N. (2022). Is quantum advantage the right goal for quantum machine learning? In PRX Quantum, 3(3). https://doi.org/10.1103/PRXQuantum.3.030101

Schuld, M., Sinayskiy, I., & Petruccione, F. (2014). The quest for a quantum neural network. In Quantum Information Processing, 13(11). https://doi.org/10.1007/s11128-014-0809-8

Shalev-Shwartz, S., Singer, Y., Srebro, N., & Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for SVM. Mathematical Programming, 127(1). https://doi.org/10.1007/s10107-010-0420-4

Singh, H., Majumder, S., & Mishra, S. (2023). Benchmarking of different optimizers in the variational quantum algorithms for applications in quantum chemistry. The Journal of Chemical Physics, 159(4). https://doi.org/10.1063/5.0161057

Spall, J. C. (1998). An overview of the simultaneous perturbation method for efficient optimization. Johns Hopkins Apl Technical Digest, 19(4), 482–492.

Suryotrisongko, H. (2020). Botnet DGA Dataset. IEEE Dataport. https://doi.org/10.21227/rg6z-z622

Suryotrisongko, H., & Musashi, Y. (2021). Evaluating hybrid quantum-classical deep learning for cybersecurity botnet DGA detection. Procedia Computer Science, 197. https://doi.org/10.1016/j.procs.2021.12.135

Suryotrisongko, H., & Musashi, Y. (2022). Hybrid quantum deep learning and variational quantum classifier-based model for botnet DGA attack detection. International Journal of Intelligent Engineering and Systems, 15(3). https://doi.org/10.22266/ijies2022.0630.18

Suryotrisongko, H., Musashi, Y., Tsuneda, A., & Sugitani, K. (2022). Robust botnet DGA detection: Blending XAI and OSINT for cyber threat intelligence sharing. IEEE Access, 10.https://doi.org/10.1109/ACCESS.2022.3162588

Tehrani, M., Sultanow, E., Buchanan, W. J., Amir, M., Jeschke, A., Chow, R., & Lemoudden, M. (2023). Enabling Quantum Cybersecurity Analytics in Botnet Detection: Stable Architecture and Speed-up through Tree Algorithms. arXiv: https://arxiv.org/abs/2306.13727

Tehrani, M. G., Sultanow, E., Buchanan, W. J., Amir, M., Jeschke, A., Houmani, M., Chow, R., & Lemoudden, M. (2024). Stabilized quantum-enhanced SIEM architecture and speed-up through Hoeffding tree algorithms enable quantum cybersecurity analytics in botnet detection. Scientific Reports, 14(1), 1732–1732. https://doi.org/10.1038/s41598-024-51941-8

Tuan, T. A., Long, H. V., & Taniar, D. (2022). On detecting and classifying DGA botnets and their families. Computers and Security, 113. https://doi.org/10.1016/j.cose.2021.102549

Tuptuk, N., & Hailes, S. (2018). Security of smart manufacturing systems. Journal of Manufacturing Systems, 47. https://doi.org/10.1016/j.jmsy.2018.04.007

Ullah, I., Khan, N., & Aboalsamh, H. A. (2013). Survey on botnet: Its architecture, detection, prevention and mitigation. 2013 10th IEEE International Conference on Networking, Sensing and Control, ICNSC 2013. https://doi.org/10.1109/ICNSC.2013.6548817

Zago, M., Gil Pérez, M., & Martínez Pérez, G. (2020). UMUDGA: A dataset for profiling DGA-based botnet. Computers and Security, 92. https://doi.org/10.1016/j.cose.2020.101719

Zhao, H., Chang, Z., Bao, G., & Zeng, X. (2019). Malicious Domain Names Detection Algorithm Based on N-Gram. Journal of Computer Networks and Communications, 2019, 1–9. https://doi.org/10.1155/2019/4612474

Zlokapa, A., Neven, H., & Lloyd, S. (2021). A quantum algorithm for training wide and deep classical neural networks. arXiv: https://doi.org/10.48550/arxiv.2107.09200