Quantum Deep Hedging

El Amine Cherrat[1,2], Snehal Raj[1], Iordanis Kerenidis[1,2], Abhishek Shekhar[3], Ben Wood[3], Jon Dee[3], Shouvanik Chakrabarti[4], Richard Chen[4], Dylan Herman[4], Shaohan Hu[4], Pierre Minssen[4], Ruslan Shaydulin[4], Yue Sun[4], Romina Yalovetzky[4], and Marco Pistoia[4]

[1]QC Ware
[2]Université de Paris, CNRS, IRIF
[3]Quantitative Research, JPMorgan Chase
[4]Global Technology Applied Research, JPMorgan Chase

Quantum machine learning has the potential for a transformative impact across industry sectors and in particular in finance. In our work we look at the problem of hedging where deep reinforcement learning offers a powerful framework for real markets. We develop quantum reinforcement learning methods based on policy-search and distributional actor-critic algorithms that use quantum neural network architectures with orthogonal and compound layers for the policy and value functions. We prove that the quantum neural networks we use are trainable, and we perform extensive simulations that show that quantum models can reduce the number of trainable parameters while achieving comparable performance and that the distributional approach obtains better performance than other standard approaches, both classical and quantum. We successfully implement the proposed models on a trapped-ion quantum processor, utilizing circuits with up to 16 qubits, and observe performance that agrees well with noiseless simulation. Our quantum techniques are general and can be applied to other reinforcement learning problems beyond hedging.

1 Introduction

In financial markets, hedging is the important activity of trading with the aim of reducing risk. For example, buyers and sellers of derivative contracts will often trade the asset underlying the derivative in order to mitigate the risk of adverse price movements. Classical financial mathematics provides optimal hedging strategies for derivatives in idealized friction-less markets, but for real markets these strategies must be adapted to take into account transaction costs, market impact, limited liquidity, and other constraints.

Finding optimal hedging strategies in the presence of these important real-world effects is highly challenging. Deep Hedging [1, 2] is a framework for the application of modern reinforcement learning techniques to solve this problem. One starts by defining a reinforcement learning environment for the hedging problem and a trading goal of maximizing a risk-adjusted measure of cumulative future returns. Then, one can apply standard deep reinforcement learning algorithms, such as policy-search or actor-critic approaches,

El Amine Cherrat: el.amine-cherrat@qcware.com
Yue Sun: yue.sun@jpmorgan.com

by designing neural network architectures to model the trading strategy and by defining a training loss function to find the optimal parameters that maximize the trading goal.

Beyond Deep Hedging, the applicability of machine learning to finance has grown significantly in recent years as highly efficient machine learning algorithms have evolved over time to support different data types and scale to larger data sets. For instance, supervised learning can be used for asset pricing or portfolio optimization [3, 4], unsupervised learning for portfolio risk analysis and stock selection [5, 6], and reinforcement learning for algorithmic trading [7, 8]. At the same time, machine learning has been identified as one of the most important domains of applicability of quantum computing given the potential ability of quantum computers to solve classically-intractable computational problems [9], perform linear algebraic operations efficiently [10], compute gradients [11] and provide variational-type approaches [12]. Such techniques have already been considered for financial use cases [13–18], and in fact finance is estimated to be one of the first industry sectors to benefit from quantum computing [19, 20].

In this work, we develop quantum deep learning methods and show how they can be a powerful tool for Deep Hedging. Quantum deep learning methods, and, in particular, quantum neural networks based on parametrized quantum circuits, have been proposed as a way to enhance the power of classical deep learning [12]. Such quantum neural networks may in general be difficult to train, often encountering problems of barren plateaus or vanishing gradients [21]. Here, we design quantum neural layers based on Hamming-weight preserving unitaries constructed out of 2-dimensional rotation gates (called RBS gates), and prove that they can be efficiently trainable, in the sense that the variance of the gradients decays only polynomially with the number of qubits. These quantum layers, first defined in [22], naturally provide models that have orthogonal features, improving interpretability [23], allowing for deeper architectures, and providing theoretical and practical benefits in generalization [24]. Depending on the input data encoding, one can control the size of the Hilbert space these neural networks are exploring while training. We call these two types of layers *orthogonal layers* and *compound layers* respectively.

First, using our orthogonal layers within classical neural network architectures, we design novel quantum neural networks for time-series. To evaluate the behavior of our quantum neural networks, we use the same example as in [1], where the market was simulated using Geometric Brownian Motion (GBM) with a single hedging instrument (equity). We then benchmark four different neural network architectures (Feed-forward, Recurrent, LSTM, Transformer), with both classical layers and our quantum layers, using the policy-search Deep Hedging algorithm [1, 2]. Our quantum neural networks achieve comparable scores as their classical counterparts while obtaining qualitatively different solutions, providing models with orthogonal features and considerably fewer parameters. It is conceivable that similar parameter reduction may be obtained by purely classical techniques such as pruning.

Second, we design a novel quantum-native reinforcement learning method for Deep Hedging. We start by formulating a quantum encoding of the environment and trading goals for Deep Hedging. We then introduce a distributional actor-critic reinforcement learning algorithm in combination with quantum neural network architectures based on compound layers for the policy and value function. Our approach is inspired by classical distributional reinforcement learning wherein the critic does not only learn the expectation of cumulative returns, but also approximates their distribution. Recent studies, such as AlphaTensor [25], have demonstrated that distributional reinforcement learning can lead to better models compared to standard reinforcement learning methods, despite the increased difficulty in training [26]. A distributional actor-critic method for Deep Hedging had

not been studied in the classical case before. We note that our quantum distributional reinforcement learning algorithms can be used more generally for reinforcement learning problems and is not limited to Deep Hedging.

Quantum computers are naturally suited to distributional reinforcement learning. Each quantum circuit explicitly encodes a mapping between exponential size distributions, and the measurement of a quantum circuit results in a sample from such a distribution. These samples can be used to simply learn the expectation of the entire distribution or to flexibly obtain extra information about the distribution, such as the expectations restricted to relevant subsets of the total range. In the case of Deep Hedging, we parametrize the value function using quantum neural networks with compound layers, which preserve the Hamming-weight subspaces of their input domain. This choice is particularly suited to the Deep Hedging setting, where when we encode a stochastic path as a binary string of up or down jumps, then it is intuitively natural that the number of net *jumps*, namely the Hamming weight of the encoding, is a major component that determines its behavior. The restriction to compound layers further makes the neural network architecture shallower and trainable.

Confirming this intuition, the quantum policies trained using our distributional actor-critic algorithm outperform those trained with policy-search based or standard actor-critic models where only the expectation of the value function is learned. These results are achieved again for a variant of the example from [1], where we used a discretized Geometric Brownian Motion as a market model both without and with transaction costs.

Last, we evaluated our framework on the Quantinuum H1-1 and H1-2 trapped-ion quantum processors [27]. In particular, we performed inference on the quantum hardware using two sets of Quantum Deep Hedging models which were classically pre-trained. First, we used the policy-search based algorithm with the LSTM and Transformer architectures instantiated with 16-qubit orthogonal layers. Second, we used the novel distributional actor-critic algorithm instantiated with compound neural networks using up to 12 qubits. We observed close alignment between noiseless simulation and hardware experiments, with our distributional actor-critic models again providing best performance. We note that some of the circuits used to instantiate our framework may be classically simulatable with only a polynomial overhead (including the setup used in our numerical experiments) [28, 29]. Nevertheless, this does not hold true for the Quantum Deep Hedging framework, which is more general and can be applied with any quantum layers. For example, it can be shown that with suitable input states which are still very easy to create, circuits used in our work become classically hard to simulate [30].

The rest of the paper is organized as follows. Section 2 introduces preliminaries for quantum computing and reinforcement learning. In Section 3, the problem of Deep Hedging is formulated and policy-search and actor-critic algorithms are presented. Section 4 presents our orthogonal and compound neural networks and proves their trainability. Section 5 introduces a novel quantum framework for reinforcement learning and applies it to Deep Hedging. Section 6 reports our simulation and hardware implementation results. Finally, Section 7 concludes with remarks and open questions.

2 Preliminaries

2.1 Quantum Computing

Quantum computing [31] is a new paradigm for computing that uses the postulates of quantum mechanics to perform computation. The basic unit of information in quantum

computing is a qubit. The state of a qubit can be written as:

$$|x\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle,$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$ and $|\alpha_0|^2+|\alpha_1|^2 = 1$, and corresponds to a unit vector in the Hilbert space $\mathcal{H} = \text{span}_{\mathbb{C}}\{|0\rangle, |1\rangle\}$. A qubit can be generalized to n-qubit states, which are represented by unit vectors in $\mathcal{H}^{\otimes n} \simeq \mathbb{C}^{2^n}$. Specifically,

$$|x\rangle = \sum_{b\in\{0,1\}^n} \alpha_b|b\rangle,$$

with $\alpha_b = \langle x|b\rangle \in \mathbb{C}$ and $\sum_{b\in\{0,1\}^n} |\alpha_b|^2 = 1$, where $\{|b\rangle \mid b \in \{0,1\}^n\}$ is the computational basis of $\mathcal{H}^{\otimes n}$. Quantum states evolve through the application of unitary operators, which are $2^n \times 2^n$ unitary matrices. Applying a unitary operator U to an n-qubit state $|x\rangle$ results in a new quantum state $U|x\rangle$.

Quantum states can be measured, and the measurement process reveals information about the state of the system. The probability of a quantum state $|x\rangle$ yielding outcome b from a measurement in the computational basis is given by $|\alpha_b|^2$. More generally, the measurement process is described as an observable, which is a Hermitian operator that acts on the quantum state. The observable is given by $O = \sum_m \mathrm{o}_m P_m$, where o_m are real numbers that specify the measurement outcomes and P_m are projection operators onto the subspaces that correspond to each outcome. This can be calculated as the inner product between the state and the corresponding projection operator, or $p_m(x) = \langle x|P_m|x\rangle$. The expectation of measuring the observable O in the state $|x\rangle$ is defined as the sum of the measurement outcomes weighted by their corresponding probabilities, or $\sum_m \mathrm{o}_m p_m(x)$. This expectation can also be written as the trace of the observable O and the density matrix $\rho(x) = |x\rangle\langle x|$, i.e., $\langle x|O|x\rangle = \mathrm{Tr}\left[O\rho(x)\right]$, where Tr is the trace operator. The trace operation returns the sum of the diagonal elements of the matrix, which corresponds to the expected value of the measurement outcome in the state $|x\rangle$.

2.2 Reinforcement Learning

The aim of reinforcement learning [32] is to train an agent to discover the policy that maximizes the agent's performance in terms of cumulative future reward. And while interacting with the environment, the agent only receives a reward signal. The agent can take an action from a set of possible actions based on a policy that maps each state to an action.

Environments in reinforcement learning are modeled as decision making problems defined by specifying the state set, the action set, the underlying model describing the dynamics of the environment, and the reward mechanism. The usual framework used to describe the environment's elements in reinforcement learning are *Markov Decision Processes* (MDPs). In this paper, we will consider finite-horizon MDPs that can be defined as follows:

Definition 1 (Finite-horizon MDP)**.** *A finite-horizon MDP $\mathcal{M}$ is defined by a tuple $(\mathcal{S}, \mathcal{A}, p, r, T)$, where $\mathcal{S}$ is is the state space, $\mathcal{A}$ is the action space, $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function with $\Delta(\mathcal{S})$ the set of distributions over $\mathcal{S}$, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and $T \in \mathbb{N}^*$ is the time horizon.*

Starting from a state $s_t \in \mathcal{S}$, a single interaction with the environment can be represented by a sequence of actions $\{a_{t'}^{\pi}\}_{t'=t}^{T}$ selected based on a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$,

and a sequence of random states $\{s_{t'}\}_{t'=t}^{T}$ that follow the MDP transitions p. The cumulative return R_t^π is the sum of rewards from time-step t to T and is given by

$$R_t^\pi(s_t, s_{t+1}, \ldots, s_T) = \sum_{t'=t}^{T} r(s_{t'}, a_{t'}^\pi).$$

In reinforcement learning, the objective is to find the policy π^* that maximizes the expected return for all states s_t. The expected value of the return, taking into consideration all possible future states $\{s_{t'}\}_{t'=t}^{T}$ resulting from the environment transitions described by p, is referred to as the value function v^π. For any time-step t, and denoting by $\boldsymbol{s}_{t'} \in \Delta(\mathcal{S})$ the random variable that takes values in $\mathcal{S}$ according to the environment dynamics for $t' > t$ and knowing $\boldsymbol{s}_t = s_t$, the cumulative return is defined as follows:

$$v_t^\pi(s_t) = \mathbb{E}[R_t^\pi(\boldsymbol{s}_t, \boldsymbol{s}_{t+1}, \ldots, \boldsymbol{s}_T)|s_t].$$

Typically, the goal of reinforcement learning is to find a policy that maximizes the expected return, and different algorithms have been developed to achieve such objectives [33]. The value function is used to evaluate policies in order to find the one that maximizes the expected return.

3 Deep Hedging Formulation

3.1 Deep Hedging Environment

Deep Hedging is a classical algorithm that treats hedging of a set of derivatives as a reinforcement learning problem. This algorithm was first introduced in [1, 2] and has been further developed in subsequent works such as [34–37]. In the original approach, the authors use a reinforcement learning environment associated with Deep Hedging that employs finite-horizon MDPs where there is a different state and action space per time-step. The time horizon T represents the maximum maturity of all instruments, and $\mathcal{S}_t$ and $\mathcal{A}_t$ are the sets of observed market states and available actions at time-step t, respectively.

During the interaction with the environment, the agent observes a market state $s_t \in \mathcal{S}_t$ that contains all current and past market information (prices, cost estimates, news, internal state of neural networks, ...), and takes an action

$$a_t^\pi = \pi_t(s_t) \in \mathcal{A}_t,$$

potentially subject to constraints (liquidity limits, risk limits, ...), according to a deterministic policy $\pi := \{\pi_t\}_{t=0}^{T}$. Then, the environment transitions to the next state s_{t+1}, according to $p_t : \mathcal{S}_t \to \Delta(\mathcal{S}_{t+1})$, that is assumed not to depend on the action a_t^π since actions have no market impact in the Deep Hedging model [37]:

$$p_t(s_{t+1}|s_t) := \mathbb{P}[s_{t+1}|s_t].$$

Subsequently, the agent receives a total reward of

$$r_t^\pi(s_t) := r_t(s_t, a_t^\pi) = r_t^+ - r_t^-,$$

where r_t^+ is the source of positive rewards such as the generated cashflows and r_t^- represents the source of negative rewards and corresponds to the transaction costs. The interaction ends after a terminal state $s_T \in \mathcal{S}_T$ is reached. The cumulative sum of the rewards perceived during this interaction can be rewritten as

$$R_t^\pi(s_T) \equiv R_t^\pi(s_t, s_{t+1}, \ldots, s_T),$$

since, by definition, s_T contains all previous states $s_{t'}$ for all $t \leq t' < T$.

3.2 Trading Goals in Deep Hedging

The standard objective in reinforcement learning problems is to find the optimal strategy π^* that maximizes the value function v^π over all policies π. As discussed in Section 2.2, the value function is usually defined as the expected cumulative return, which, in this context, would be $\mathbb{E}[R_t^\pi(\boldsymbol{s}_T)|s_t]$. However, in order to take into account the inherent risk in trading strategies, the goal of Deep Hedging is to find a deterministic optimal policy π^* that maximizes the value function defined, for some policy π, as

$$v_t^\pi(s_t) = \mathrm{E}\left[R_t^\pi(\boldsymbol{s}_T)|s_t\right],$$

where E is the expected utility defined by a risk-averse (i.e., concave) utility function.

Various forms of utility functions have been proposed that satisfy the concave requirement. For a more detailed discussion on the desired properties of the utility function and examples of commonly used forms, see [1, 2]. In this paper, we use the exponential utility function E_λ, which is an example of a monetary utility function that is increasing, concave, and cash-invariant [37]. Specifically, it is defined for some risk aversion level $\lambda > 0$ as

$$\mathrm{E}_\lambda[\boldsymbol{X}] := -\frac{1}{\lambda}\log\mathbb{E}[\exp(-\lambda\boldsymbol{X})].$$

The parameter λ can be used to reflect the investor's risk tolerance, with larger values indicating more risk aversion. With this exponential utility function, the value function v^* associated with the optimal policy π^* is given by

$$\forall t,\ \forall s_t \in \mathcal{S}_t, \quad v_t^*(s_t) = \sup_\pi v_t^\pi(s_t) = -\frac{1}{\lambda}\log\left\{\inf_\pi \mathbb{E}\left[\exp\left(-\lambda R_t^\pi\left((\boldsymbol{s}_T)\right)|s_t\right]\right\}.$$

Since the Deep Hedging objective is formulated in terms of risk-adjusted measures, the value function is no longer a solution to the standard Bellman equation. However, conventional reinforcement learning algorithms can be adapted to find policies that maximize the utility. Two algorithms have been developed to solve the Deep Hedging problem. The first approach, called policy-search Deep Hedging [1, 2], uses a neural network to model the policy and updates the set of parameters using gradient descent to minimize the policy loss function. The second approach, actor-critic Deep Hedging [37], represents both the policy and the value function with neural networks and computes the utility using the policy function to update the value network, which is then used to update the policy network.

4 Quantum Neural Networks with Orthogonal and Compound Layers

A Quantum Neural Network (QNN) consists of a composition of parametrized unitary operations, whose parameters can be trained to provide machine learning models for classification or regression tasks. While current quantum hardware is still far from being powerful enough to compete with classical machine learning algorithms, many interesting quantum machine learning algorithms have started to appear, such as for regression [38, 39], classification [40–42], generative modeling [43, 44], and reinforcement learning [45].

In general, a QNN consists of data-loading layers and trainable layers, which are both parametrized unitary operations. In some architectures, the data-loading is an explicit-encoding scheme that is used to directly embed the classical input data into the amplitudes of a quantum state. While in others, these parts only implicitly encode the data and prepare a state whose amplitudes are some complex non-linear function of the input data. The latter is known in literature as a quantum feature map [46]. After the unitary operators

are applied, the resulting quantum state is probed to produce classical data that can be used for inference or training.

One popular approach, based on variational quantum circuits [12], is to apply an alternating sequence of quantum feature maps and trainable parts and output the expectation of the resulting state with respect to some observable [47, 48]. A second class of architectures encodes in the amplitudes the input data and performs trainable unitary operations that reproduce the linear layers of certain classical neural networks but with reduced computational complexity [22, 49]. The output is obtained through quantum-state tomography and non-linear operations are then applied classically. After applying the non-linearity, the data is reloaded onto a quantum state, and the process is repeated to compose layers.

Such quantum circuits can be trained using classical gradient descent methods until convergence. For variational quantum circuits, where the output is an expectation value, the gradient can be computed using the parameter-shift rule [38, 50]. One needs to be very careful in designing such quantum neural networks, since they may in general be difficult to train, often encountering problems, such as, barren plateaus or vanishing gradients [21].

In Sections 4.1 and 4.2 below we review two different types of quantum layers built from Hamming-weight preserving unitaries and that can be used to provide a natural quantization of classical neural network architectures. In Section 4.3 we discuss neural networks architectures that make use of these layers. While similar techniques have appeared previously in [42, 49], our discussion is more systematic and extends the techniques to a larger set of architectures. Finally, in Section 4.4 we discuss the properties of quantum neural networks with compound layers and prove their trainability.

4.1 Quantum Orthogonal Layers

The quantum orthogonal layer was proposed by Kerenidis et al. [22] to simulate traditional orthogonal layers with reduced complexity at inference time. Specifically, a quantum orthogonal layer on n-qubits acts as an element of $\mathrm{SO}(n)$ when restricted to the span of the computational basis states with Hamming-weight one, i.e. the unary basis. This is achieved by composing two-qubit Reconfigurable Beamsplitter (RBS) gates. An RBS gate acting on the i-th and j-th qubits implements a Givens rotation:

$$\mathrm{RBS}_{ij}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

If the goal is to apply an orthogonal matrix to classical data in a vector $\boldsymbol{x} \in \mathbb{R}^n$, then $\boldsymbol{x}$ can be efficiently amplitude encoded in a quantum state in the unary basis with a log-depth circuit [51, 52]. The unary data loader (depicted in Figure 1) uses n qubits and maps the all-zeros basis state $|0\rangle^{\otimes n}$ to the state $|\boldsymbol{x}\rangle$ as follows:

$$U_L(\boldsymbol{x}) : |0\rangle^{\otimes n} \to |\boldsymbol{x}\rangle = \frac{1}{\|\boldsymbol{x}\|} \sum_{i=1}^{n} x_i |e_i\rangle,$$

where $\|\cdot\|$ represents the ℓ_2 norm and $|e_i\rangle$ is the i^{th} unary basis quantum state represented by $|0\rangle^{\otimes(i-1)}|1\rangle|0\rangle^{\otimes(n-i)}$.

Let $G(i, j, \theta)$ denote the Givens rotation applied to the i-th and j-th unary basis vector, i.e. e_i and e_j, $\boldsymbol{\theta}$ a vector of angles, and $\mathcal{T}$ is a list of triplets (i, j, m). The orthogonal layer is defined by

$$U(\boldsymbol{\theta}) = \prod_{(i,j,m)\in\mathcal{T}} \mathrm{RBS}_{ij}(\theta_m).$$

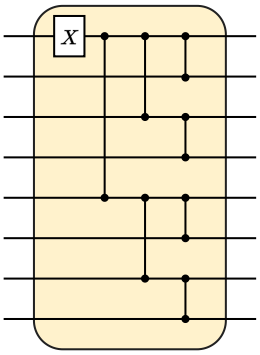
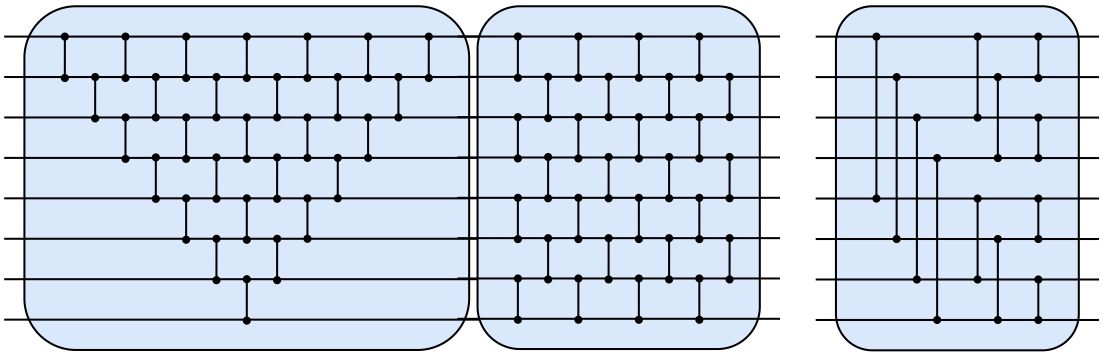


Figure 1: A quantum circuit with logarithmic depth for data loading. Vertical lines represent RBS gates with parameters that are dependent on the input $\boldsymbol{x}$. The unitary represented by this data loader is denoted as $U_L(\boldsymbol{x})$.



(a) Pyramid architecture (b) Brick architecture (c) Butterfly architecture

Figure 2: Various Hamming-weight preserving circuits used in quantum orthogonal layers. These circuits are parameterized by a set of parameters $\boldsymbol{\theta}$, with each parameter representing the angle of a specific RBS gate. The parameterized unitary represented by this layer is expressed as $U(\boldsymbol{\theta})$.

It acts as $U(\boldsymbol{\theta})|\boldsymbol{x}\rangle = W|\boldsymbol{x}\rangle$, where $W = \prod_{(i,j,m)\in\mathcal{T}} G(i,j,\theta_m)$. Since the dimension of the Hamming-weight one subspace is n for n qubits, there exist efficient quantum-state tomography procedures for reading out the resulting quantum state encoding the matrix-vector product $W|\boldsymbol{x}\rangle$ [22].

The fact that circuits of RBS can only span elements of $\mathrm{SO}(n)$ avoids the computational overhead that is associated with the need to re-orthogonalize the weight matrix in the classical case. This can be implemented with a linear-depth quantum circuit. Note that the application of each such layer can also be performed classically in time $\mathcal{O}(n^2)$. Furthermore, orthogonal layers are efficiently trainable, as the dimension of the space they explore is linear in the number of qubits used. Specifically, these layers are trained by classically simulating the circuit and using quantum for inference.

There are different linear-depth circuits for $U(\boldsymbol{\theta})$, highlighted in Figure 2, each with its own unique properties. The Pyramid architecture, as described in [22], consists of $n(n-1)/2$ RBS gates arranged in a pyramid-like structure and has a linear depth. This architecture allows for the representation of all possible orthogonal matrices of size $n \times n$. The Brick architecture is a variation of the Pyramid architecture and also consists of $n(n-1)/2$ RBS gates. However, it has a more compact layout of gates, while still exhibiting similar properties. Both the Pyramid and Brick architectures can be implemented in hardware with nearest-neighbour connectivity between qubits.

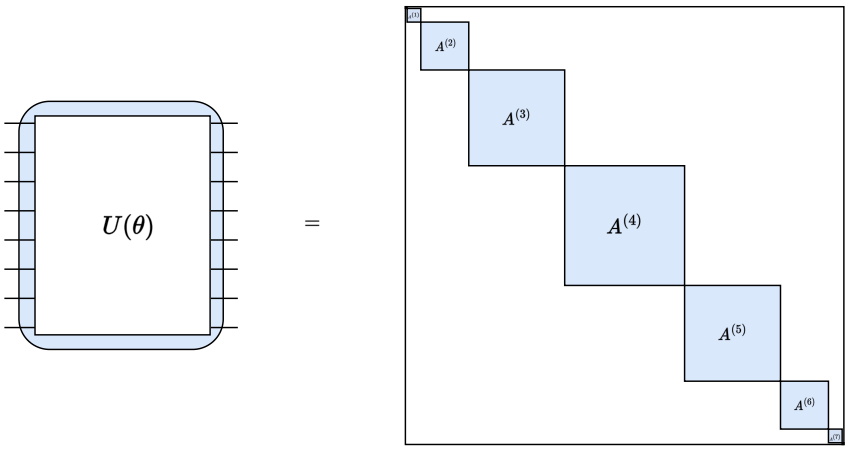


Figure 3: A quantum compound layer $U(\boldsymbol{\theta})$ acts as a block diagonal unitary on each fixed Hamming-weight subspace.

On the other hand, the Butterfly architecture, which was proposed in [49], uses logarithmic depth circuits with a linear number of gates to implement a quantum orthogonal layer. This architecture requires all-to-all connectivity in the hardware layout. To summarize, an orthogonal layer with input size n uses a parametrized quantum circuit with n qubits and a number of parameters equal to $\binom{n}{2}$ for a Pyramid or Brick circuit and can be implemented on hardware with nearest-neighbour connectivity. For a Butterfly circuit, the number of parameters is $\frac{n}{2}\log(n)$ and requires all-to-all connectivity in the hardware.

Because classical data can be efficiently loaded onto a quantum state and retrieved from a quantum orthogonal layer, it is possible to compose quantum orthogonal layers with nonlinear activation functions. More specifically, after applying the sequence of RBS gates, the matrix-vector product $W|\boldsymbol{x}\rangle$ is readout and an activation function is applied classically. The resulting vector is then loaded onto a quantum state using the unary data loader for the next layer. In Section 4.3, we will make use of this scheme to construct various quantizations of classical neural architectures for time series.

4.2 Quantum Compound Layers

The quantum compound layer is a natural and powerful generalization of the orthogonal layer [42] and a version of it has been previously used [49] to implement quantum analogues of vision transformers. The prefix "compound" refers to the fact that the quantum circuits implement linear operators on the exterior power of a vector space.

For an n-dimensional vector space V with orthonormal basis $\{e_i\}_{i=1}^n$, the k-th exterior power $\bigwedge^k V$ is the $\binom{n}{k}$-dimensional vector space spanned by the k-fold alternating products of vectors in V:

$$\bigwedge^k V := \text{span}_{\mathbb{C}}\{e_{i_1} \wedge e_{i_2} \wedge \ldots e_{i_k} | i_1 < i_2 < \cdots < i_k \in [n]\}.$$

The alternating property implies that for any permutation σ of the indices:

$$e_{\sigma(i_1)} \wedge e_{\sigma(i_2)} \wedge \cdots \wedge e_{\sigma(i_k)} = \text{sign}(\sigma) \times e_{i_1} \wedge e_{i_2} \wedge \cdots \wedge e_{i_k}.$$

The direct sum $\bigoplus_{k=0}^n \bigwedge^k V$ equipped with the alternating product forms the exterior algebra of V, denoted $\bigwedge V$.

For any linear operator A on V, there exists an extension to a linear operator $A^{(k)}$ on $\bigwedge^k V$, which acts as

$$A^{(k)}(e_{i_1} \wedge e_{i_2} \wedge \cdots e_{i_k}) = Ae_{i_1} \wedge Ae_{i_2} \wedge \cdots \wedge Ae_{i_k}$$

on k-vectors. The matrix for the extended operator, called the k-th (multiplicative) compound matrix, has as entries $A^{(k)}_{IJ} = \det(A_{IJ})$, where I and J are k-sized subsets of the rows and columns of A, respectively. Furthermore, there exists a unique linear operator $\mathcal{A} := \bigoplus_{k=0}^{n} A^{(k)}$ over $\bigwedge V$ such that the restriction to the k-th exterior power is $A^{(k)}$.

In the quantum setting, the k-vectors $e_{i_1} \wedge \cdots \wedge e_{i_k}$ are mapped to computational basis states $|S\rangle$, where $S \in \{0,1\}^n$, $|S| = k$, and $\forall t \in [k], S_{i_t} = 1$. Thus n-qubits can be used to encode a projectivization of the exterior algebra $\bigwedge V$. To apply compound matrices to k-vectors in the qubit encoding, we utilize Fermionic Beam Splitter (FBS) gates, whose action on qubits i and j depends on the parity of the qubits between i and j. On a computational basis state $|S\rangle$, the FBS gate acts on qubits i and j as the unitary matrix below:

$$\mathrm{FBS}_{ij}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & (-1)^{f(i,j,S)} \sin(\theta) & 0 \\ 0 & (-1)^{f(i,j,S)+1} \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where $\theta \in [0, 2\pi)$ and $f(i,j,S) = \sum_{i<k<j} s_k$, and as identity on all other qubits. An FBS gate can be implemented as the composition of controlled-X gates, controlled-Z gates, and RBS gates.

Like in the previous subsection, let $G(i,j,\theta)$ denote the Givens rotation applied to the i-th and j-th basis vector, i.e. e_i and e_j, $\boldsymbol{\theta}$ a vector of angles, and $\mathcal{T}$ is a list of triplets (i,j,m). The quantum compound layer is defined by

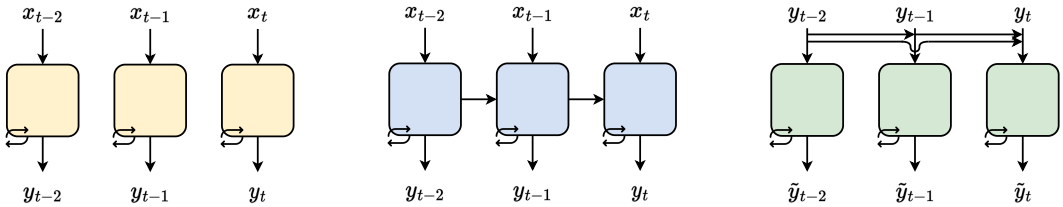
$$U(\boldsymbol{\theta}) = \prod_{(i,j,m)\in\mathcal{T}} \mathrm{FBS}_{ij}(\theta_m).$$

It can be shown [53] that this layer acts as $U(\boldsymbol{\theta})|S\rangle = A^{(k)}|S\rangle$ for $|S| = k$, where $A^{(k)}$ is the k-th multiplicative compound of $A = \prod_{(i,j,m)\in\mathcal{T}} G(i,j,\theta_m)$. Thus the operation U acts as $\mathcal{A}$ over the quantum state space, in other words it is a block diagonal unitary that acts separately on each fixed Hamming-weight subspace (see Figure 3).

The compound layer is similar to the circuits we described in the orthogonal layer case, where there, given we only consider the unary basis, the FBS gates can be replaced by the simpler RBS gates. Note as well, that RBS and FBS gates acting on nearest-neighbor qubits, as in the Pyramid and Brick circuits, are also equivalent. The main difference in the compound layer comes from the fact the data loading part is not restricted to the unary basis, and thus one needs to consider the entire exponential size block-diagonal unitary, and not only its linear size restriction to the unary basis.

Thus, one can see that by controlling the Hamming-weight of the basis states used in the data loading part one can smoothly control the size of the explored space, from linear size, when using a unary basis for data loading, to an exponential size, when we use all possible Hamming-weights, as is the case for example when we load each coordinate of a classical data point by performing a one-qubit rotation with an appropriate data-dependent angle.

Lastly, since any element of $\mathrm{SO}(n)$ can be expressed as a product of $\mathcal{O}(n^2)$ Givens rotations, the direct sum of compound matrices can be implemented efficiently as a composition of FBS gates. Thus quantum computation can be used to efficiently parametrize and apply compound matrices over the exterior algebra.



(a) Feed-forward architecture. (b) Recurrent architecture. (c) Attention mechanism.

Figure 4: Diverse quantum neural network architectures for time-series data, featuring orthogonal layers in each block as outlined in Section 4.3. Here, $\boldsymbol{x}_t$ and $\boldsymbol{y}_t$ denote the time-series input and output, respectively, while $\tilde{\boldsymbol{y}}_t$ represents the output after being adjusted by the attention mechanism.

4.3 Quantum Neural Network Architectures with Orthogonal Layers

Our aim is to develop quantum neural networks capable of processing sequential data. To achieve this, we will utilize classical neural network architectures that have proven to be effective in dealing with time-series data. However, we will extend these classical architectures by replacing the linear layers with our orthogonal layers. This approach offers an alternative to the use of quantum variational circuits, which are commonly used in quantum machine learning. As discussed in Section 4.1 orthogonal layers use only the unary basis whose size is equal to the number of qubits, and hence one can easily perform tomography to obtain a classical description of the output and apply a nonlinearity. By combining the strengths of classical neural networks with the unique properties of quantum orthogonal layers, we hope to achieve improved results in processing sequential data.

We designed quantum neural networks to process input time-series data $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ and produce the final output sequence $(\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_T)$. We split these architectures into two categories: feed-forward and recurrent architectures.

For the purpose of details, we assume that the input and output have the same dimension n and that this dimension is maintained across layers. Additionally, these architectures are made up of blocks that can be repeated to create deeper architectures. Here, we will assume that the number of blocks is one.

- **Feed-forward Architectures:** A classical Feed-forward neural network consists of multiple layers of transformations, where information flows from input to output without looping back. In each layer, a linear transformation, bias shift, and a non-linear function are applied. The output is calculated as

$$\boldsymbol{x}_t = \mathrm{f}\left(W\boldsymbol{x}_t + \beta\right),$$

  where f is the activation function, and W and β are the weights and biases. The number of parameters in each layer of a classical network is $\mathcal{O}(n^2)$. Our proposed quantum equivalent (Figure 4a) calculates each transformation as

$$\boldsymbol{y}_t = \mathrm{f}\left(\gamma \circ U(\boldsymbol{\theta})|\boldsymbol{x}_t\rangle + \beta\right),$$

  where $\circ$ represents the element-wise product, $U(\boldsymbol{\theta})|\boldsymbol{x}_t\rangle$ is the output of a quantum orthogonal layer retrieved using tomography, γ is a scaling factor used to rescale each feature, β is a shift that acts as the bias, and f is a non-linear function such as sigmoid, tanh, or ReLU. All parameters $\boldsymbol{\theta}$, γ, and β are trainable and shared across the networks used at different time steps. The total number of trainable parameters

is $\mathcal{O}(n^2)$ when using the Brick and Pyramid architectures, and $\mathcal{O}(n\log(n))$ for the Butterfly architecture. Additionally, the parameters of the quantum orthogonal layers can either be shared across layers or not, depending on the requirements of the time-series model being used.

- **Recurrent Architectures:** Recurrent neural networks are designed to handle sequential data. One example of a recurrent architecture is the standard Recurrent Neural Network (RNN). In this example, we will show how to provide a quantum version of RNNs (Figure 4b), but the same approach can be applied to other recurrent models such as the LSTM. RNNs consist of a repeating module with a hidden state, $\boldsymbol{h}_t$, that allows information to be passed from one step of the sequence to the next. At each time-step, the network takes as input $\boldsymbol{x}_t$ and the hidden state from the previous time-step, $\boldsymbol{h}_{t-1}$. The hidden state is updated as

$$\boldsymbol{h}_t = \mathrm{f}\left(W_x\boldsymbol{x}_t + W_h\boldsymbol{h}_{t-1} + \beta\right),$$

where f is an activation function, W_x and W_h are weight matrices, and β is a bias vector. The quantum analogue of this update is represented as

$$\boldsymbol{h}_t = \mathrm{f}\left(\gamma_x \circ U(\boldsymbol{\theta}_x)|\boldsymbol{x}_t\rangle + \gamma_h \circ U(\boldsymbol{\theta}_h)|\boldsymbol{h}_{t-1}\rangle + \beta\right),$$

where we now have two orthogonal layers with parameters $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_h$, one for the input state $\boldsymbol{x}_t$ and another for the previous hidden state $\boldsymbol{h}_{t-1}$. We also have two scaling factors, γ_x and γ_h. The hidden state $\boldsymbol{h}_t$ is then used to generate the output at each time-step, $\boldsymbol{y}_t$, through another layer, which can be implemented using another orthogonal layer to map it to the output. Moreover, the parameters are shared across layers and we can show that the number of trainable parameters per layer in the quantum Recurrent Neural Networks grows similarly to that in the quantum Feed-forward Neural Networks, with the total number of parameters being $\mathcal{O}(n^2)$ for the Brick and Pyramid architectures and $\mathcal{O}(n\log(n))$ for the Butterfly architecture.

We also define an attention mechanism that can be applied to the output sequence $(\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_T)$ to create a transformer architecture.

- **Attention Mechanism:** The attention mechanism, as used in the Transformer architecture [54], can be applied to both Feed-forward and Recurrent Neural Networks. Here, we describe a basic quantum attention mechanism (Figure 4c), but it can be generalized. Given an output sequence $(\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$, the goal of the attention mechanism is to compute the output $(\tilde{\boldsymbol{y}}_1, \tilde{\boldsymbol{y}}_2, \ldots, \tilde{y}_T)$ as $\tilde{\boldsymbol{y}}_t = \sum_{t'\leq t} w_{t,t'}\boldsymbol{y}_{t'}$ where the weights $w_{t,t'}$ are computed by considering all previous time steps:

$$w_{t,t'} \propto \exp(\boldsymbol{y}_{t'} W_y \boldsymbol{y}_t/\tau),$$

where W_y is a trainable attention matrix that combines the query and key matrices into one matrix, as described in [49], and τ is a temperature parameter. In the quantum case, we use

$$w_{t,t'} \propto \exp(\langle \boldsymbol{y}_{t'}|U(\boldsymbol{\theta}_y)|\boldsymbol{y}_t\rangle/\tau),$$

where γ_y is a scaling factor used to rescale each feature, $U(\boldsymbol{\theta}_y)$ are the parameters of the quantum orthogonal layer that computes the attention weights w. The dot product between $|\boldsymbol{y}_{t'}\rangle$ and $U(\boldsymbol{\theta}_y)|\boldsymbol{y}_t\rangle$ can be computed quantumly using an additional data-loader to unload $\boldsymbol{y}_{t'}$ after applying $U(\boldsymbol{\theta}_y)$ to $\boldsymbol{y}_t$. This procedure is similar to the one described in [52].

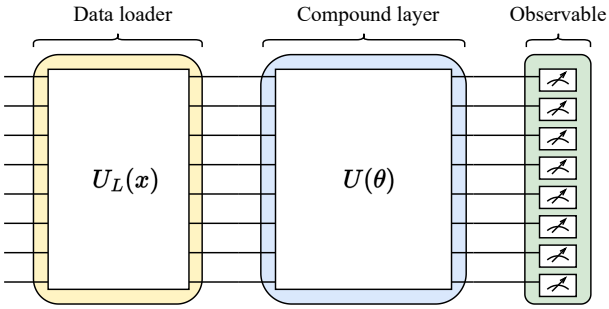


Figure 5: A quantum compound neural network. $U_L(\boldsymbol{x})$ refers to a general data loader unitary. $U(\boldsymbol{\theta})$ denotes a Hamming-weight preserving unitary as for example the ones shown in Figure 2.

4.4 Properties of Quantum Compound Neural Networks

We define a quantum compound neural network to be the standard variational QNN, i.e.

$$C(\boldsymbol{\theta}, \boldsymbol{x}) = \mathrm{Tr}[OU(\boldsymbol{\theta})\rho(\boldsymbol{x})U^\dagger(\boldsymbol{\theta})],$$

where O is an observable that preserves Hamming-weight, e.g. diagonal in the computational basis), U is a quantum compound layer, and

$$\rho(\boldsymbol{x}) = U_L(\boldsymbol{x})[|0\rangle\langle 0|]^{\otimes n}U_L^\dagger(\boldsymbol{x})$$

is a quantum feature map (Figure 5). As mentioned in Section 4.2, it is sufficient to use only RBS gates in the Brick architecture to implement a quantum compound layer. Thus, without loss of generality, we define $U(\boldsymbol{\theta})$ to consist only of RBS gates in the Brick architecture.

Under these assumptions, the output of the QNN decomposes as

$$C(\boldsymbol{\theta}, \boldsymbol{x}) = \sum_{k=0}^{n} \mathrm{Tr}[P_k\rho(\boldsymbol{x})]\mathrm{Tr}[O^{(k)}A_{\boldsymbol{\theta}}^{(k)}\rho^{(k)}(\boldsymbol{x})(A_{\boldsymbol{\theta}}^{(k)})^{-1}],$$

where P_k is the projector onto the Hamming-weight-k subspace, $O^{(k)} = P_kOP_k$, $\rho^{(k)} = P_k\rho P_k$, and $A_{\boldsymbol{\theta}}^{(k)}$ is the k-th compound matrix associated with the Givens circuit $U(\boldsymbol{\theta})$ in the manner discussed earlier. One potential application of such a subspace-preserving QNN could be the following. Suppose there is some canonical grouping of the input data $\boldsymbol{x} \in \mathcal{X}$ according to a partitioning function $f : \mathcal{X} \to [n+1]$. Then one could potentially construct a quantum-feature map or state preparation procedure such that $P_{f(\boldsymbol{x})}\rho(\boldsymbol{x})P_{f(\boldsymbol{x})} = \rho(\boldsymbol{x})$, i.e. the quantum states encoding inputs lying in different groups are embedded into different Hamming-weight subspaces. Then it follows that

$$C(\boldsymbol{\theta}, \boldsymbol{x}) = \mathrm{Tr}\left[O^{(f(x))}A_{\boldsymbol{\theta}}^{(f(x))}\rho^{(f(x))}(\boldsymbol{x})(A_{\boldsymbol{\theta}}^{(f(x))})^\mathsf{T}\right],$$

and the quantum compound neural network can potentially learn different functions over the different groups. This form of learning is a special case of group-invariant machine learning, which has also recently been explored in the quantum case [55]. Note that the parameters $\boldsymbol{\theta}$ are shared across the different functions which is beneficial for training. Furthermore, we show below that under Gaussian initialization, the variance of the gradient on each subspace does not vanish exponentially with the number of qubits. Thus quantum compound neural networks can be trained efficiently.

Classical neural networks over exterior algebras have been applied to manifold learning tasks, specifically for data that lies on Grassmannians [56]. The (n, k)-Grassmannian of V is a manifold containing all k-dimensional subspaces of V and can be embedded in the space $\bigwedge^k V$, where k-wedge products of orthogonal vectors define subspaces. When considering $A \in \mathrm{SO}(n)$, the operator $A^{(k)}$ maps between elements of the Grassmannian. While $\binom{n}{k}$ can be large, the application of $A^{(k)}$ to a Grassmannian can done by multiplying $n \times k$ and $n \times n$ matrices. However, optimizing the linear layers of the neural network while ensuring the data remains an orthogonal matrix can be computationally challenging.

Since the embedding of the (n, k)-Grassmannian into $\bigwedge^k V$ is not surjective, and in the quantum case we can apply compound matrices to the larger space $\bigwedge^k V$, the technique used classically for Grassmannian neural networks to reduce the dimension of the matrix-vector products cannot be applied to simulate the quantum case. In other words, such compound layers are inherently quantum and perform an operation that in the general case seems to take exponential time to perform classically. Nevertheless, we will show below that such compound layers remain trainable in certain settings.

In Section 5, we present a specific example of the Deep Hedging problem where grouping the inputs by Hamming-weight is natural in the quantum setting and improves the accuracy of the model.

Highly expressive QNNs are known to suffer from barren plateaus in their training landscape at initialization. This occurs when the variance of the gradient decays exponentially with the number of qubits, which makes sampling to estimate the gradient asymptotically intractable. Specifically, consider a typical QNN of the form

$$C(\boldsymbol{\theta}) = \mathrm{Tr}[OU(\boldsymbol{\theta})\rho U^\dagger(\boldsymbol{\theta})],$$

where O is an observable and U is a 2-design for the Haar measure μ on $\mathrm{SU}(2^n)$. Using known formulas for integration over the Haar measure on compact groups, it was shown [21] that $\mathrm{Var}_\mu[\partial_{\theta_i} C(\boldsymbol{\theta})] = \mathcal{O}(1/2^{2n})$.

More recent lines of work have shown that the symmetries of the parameterized quantum circuit also need to be considered and play an important role in understanding, for example, convergence [57, 58] and trainability [59] of QNNs. For trainability, it was shown that if the input state ρ lies in the invariant subspace $\mathcal{H}_k$, then the variance of the gradient is in the worst case $O(1/d_k^2)$, where $d_k = \dim \mathcal{H}_k$. Note there will also be a dependence on the initial state used. While this result was shown for SU, the asymptotics of Haar moments are similar for other classical compact Lie groups [60], such as SO. In the case of the compound layer, the subspaces $\mathcal{H}_k$ are spanned by computational basis states with Hamming-weight k. Thus if $\rho \in \mathcal{H}_k$, where k is such that $\binom{n}{k} = \mathcal{O}(\mathrm{poly}(n))$, then the variance of the gradient does not exponentially decay with growing system size for all initial states. This is at least the case when k is independent of n.

It was further conjectured in [59] [1] that the variance actually scales with the dimension of the (dynamical) Lie algebra restricted to the invariant subspace, which can be polynomial in n even when $\dim \mathcal{H}_k$ is exponential. In the case of the compound layer, the dimension of the Lie algebra of the compound matrix group for $\mathrm{SO}(n)$ is actually equal to the dimension of $\mathfrak{so}(n)$. Thus even though there are invariant subspaces whose dimension is exponential in n, the dimension of the Lie algebra grows at most quadratically in n. If proven true, this conjecture would imply that the variance does not decay exponentially on any subspace, e.g. for $k = n/2$.

[1] We note that after the writing of this manuscript, this conjecture was independently proven for certain observables [61–63].

It is possible to go beyond unproven conjectures by making some well justified assumptions about the initialization and measurement phases of the quantum compound neural network, namely that (1) The parameters are randomly initialized from centered Gaussian distributions with variance inversely proportional to the number of gates in the circuit (2) The final measurement is made in the computational basis. Specifically, we make a vector valued measurement by measuring $Z_i = I^{\otimes i-1} \otimes Z \otimes I^{\otimes n-i}$ for each qubit $i \in [1, n]$. Since the operators Z_i all commute, the order of measurement is arbitrary and the measurement is well-defined. Such a measurement allows for loss functions that are arbitrary functions of the measured bit-string. It suffices therefore to demonstrate that the gradient of the output after measuring each Z_i does not decay exponentially with the number of qubits. The overall gradient may still decay depending on the loss function used, but this would be a property of the loss itself and not of the quantum neural network. (3) The initial state is chosen to be either the uniform superposition, or the uniform superposition over all computational basis states of Hamming-weight k, where $1 < k < n$.

We now give a rigorous proof that the overall gradient does not vanish in this setting. We note that this is the setting in our numerical experiments (Section 6). We use the following theorem from [64], which we paraphrase below in the necessary form.

Theorem 1 (Paraphrased from [64, Theorem 4.2])**.** *Consider any n-qubit variational form with output function given by $C(\boldsymbol{\theta}) = Tr(O\Pi_{j=L}^{1} U_j(\theta_j)\rho_{in}\Pi_{j=1}^{L} U_j(\theta_j)^\dagger)$, where O is some n-qubit observable, and each U_j is expressible as the product of a constant number of parametrized 2-qubit gates (potentially with shared parameters). Then for any parameter θ_j it holds that*

$$\mathbb{E}_{\boldsymbol{\theta}} \left(\frac{\partial C}{\partial \theta_j} \right)^2 \geq \frac{1}{2} \left(\frac{\partial C}{\partial \theta_j} \right)^2 \Big|_{\boldsymbol{\theta}=0},$$

when each θ_j is initialized from a normal distribution $\mathcal{N}(0, \gamma^2)$ with $\gamma^2 = \mathcal{O}(1/L)$.

From Theorem 1, we have the following theorem.

Theorem 2. *Consider a variational quantum algorithm using a quantum compound layer and a final observable Z_m (where X_m, Y_m, Z_m correspond to the application of the corresponding Pauli gate on qubit m), and let the corresponding output (as a function of the parameters $\boldsymbol{\theta}$) be $C(\boldsymbol{\theta})$. We have*

$$\mathbb{E}_{\boldsymbol{\theta}}[\|\nabla C\|^2] = \Omega\left(\frac{1}{\text{poly}(n)}\right),$$

when the parameters are initialized from a normal distribution $\mathcal{N}(0, \gamma^2)$ with $\gamma^2 = \mathcal{O}(1/n^2)$, and the input state is chosen to be $\rho_0 = |\psi\rangle\langle\psi|$ where $|\psi\rangle$ is an n-qubit state representing either the uniform superposition over computational basis states, or the equal superposition of computational basis states with Hamming-weight k, for any $1 \leq k < n$.

Proof. We observe that the number of parameterized gates in a quantum compound neural network is $\mathcal{O}(n^2)$. Using Theorem 1, it suffices for our result to show, for some parameter θ_l, that

$$\frac{1}{2} \left(\frac{\partial C}{\partial \theta_l} \right)^2 \Big|_{\theta=0} = \Omega\left(\frac{1}{\text{poly}(n)}\right).$$

All the parameterized gates in the quantum compound neural network are RBS gates. Let the gate corresponding to θ_l act on the qubits i, j. The corresponding unitary is

$$\mathrm{RBS}_{ij}(\theta_l) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_l) & \sin(\theta_l) & 0 \\ 0 & -\sin(\theta_l) & \cos(\theta_l) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

It can be verified by computation that $\mathrm{RBS}_{ij}(\theta_l) = \exp(-i\theta_l H_{ij}^{\mathrm{RBS}})$ where, $H_{ij}^{\mathrm{RBS}} = \frac{Y_i \otimes X_j - X_i \otimes Y_j}{2}$. Finally define $U_-(\theta_{1:\,l-1}), U_+(\theta_{l+1:\,L})$ denote the sections of the parameterized circuit before and after the l^{th} parameterized gate. By explicit differentiation, we have

$$\left(\frac{\partial C}{\partial \theta_l}\right)\Bigg|_{\theta=0} = -i\langle\psi|U_-^\dagger[H_{ij}^{\mathrm{RBS}}, U_+^\dagger Z_m U_+]U_-|\psi\rangle\Big|_{\theta=0} = -i\langle\psi|[H_{ij}^{\mathrm{RBS}}, Z_m]|\psi\rangle.$$

If $m \neq i, j$, it is clear that $[H_{ij}^{\mathrm{RBS}}, Z_m] = 0$. We therefore consider the case when $m \in \{i, j\}$. In the rest of the proof, we assume without loss of generality that $i = 0, j = 1, m = 0$. We have,

$$[H_{01}^{\mathrm{RBS}}, Z_0] = i(Y_0 \otimes Y_1 + X_0 \otimes X_1) = i\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Consider any two computational basis states $|a\rangle, |b\rangle$. Clearly $-i\langle\psi|[H_{ij}^{\mathrm{RBS}}, Z_m]|\psi\rangle = 2$ if $a = 01x, b = 10x$ or vice-versa for some $n-2$ bit string x, and 0 otherwise. We now determine $\frac{1}{2}\left(\partial C/\partial\theta_l\right)^2|_{\theta=0}$ for different possible initial states $|\psi_0\rangle$. Suppose the initial state is the uniform superposition over all computational basis states $|\psi_0\rangle = \frac{1}{2^{n/2}}\sum_{b\in\{0,1\}^n}|b\rangle$. In this case,

$$\partial_l C_k = \langle\psi_0|\left(\sigma_X^{(i)} \otimes \sigma_X^{(j)} + \sigma_Y^{(i)} \otimes \sigma_Y^{(j)}\right)|\psi_0\rangle = \frac{1}{4}.$$

Now let the initial state be the ψ_k which is the uniform superposition over all strings of Hamming-weight k. For $2 \leq k \leq n$, we have

$$\partial_l C_k = \langle\psi_k|\left(\sigma_X^{(i)} \otimes \sigma_X^{(j)} + \sigma_Y^{(i)} \otimes \sigma_Y^{(j)}\right)|\psi_k\rangle = 2\left(\frac{2\binom{n-2}{k-1}}{\binom{n}{k}}\right)^2 = \Omega(\frac{1}{n^6}).$$

By an analogous argument for the Hamming-weight 1 subspace we find that $\partial_l C_k = \Omega(\frac{1}{n^2})$. As we have shown before, the gradients do vanish for the Hamming-weight 0 subspace. □

5 Quantum Deep Hedging

In this section, we present a quantum framework for Deep Hedging, referred to as *Quantum Deep Hedging*, where we aim to leverage the power of quantum computing to enhance the deep reinforcement learning methods introduced in [1] for solving the hedging problem. We will incorporate the use of quantum neural networks, as defined in the previous sections, and provide quantum reinforcement learning solutions to this problem. Quantum reinforcement learning involves utilizing quantum computing to improve reinforcement learning algorithms. A comprehensive survey by Meyer et al. [45] outlines various approaches for incorporating quantum subroutines in these algorithms:

- **In classical environments:** A common approach involves using quantum neural networks, such as variational quantum circuits, to represent the value function [65–68] or the policy [69, 70] in classical environments. These quantum neural networks can replace their classical counterparts in various reinforcement learning training methods, including value-based, policy-based, and actor-critic. Experiments have shown that they sometimes produce better policies or value estimates when applied to small environments, but can face trainability problems with larger environments due to the barren plateaus that occur in these cicuits (as discussed in Section 4).

- **In quantum environments:** Another approach considers the case of quantum access to the environment and aims to use this access to achieve a significant speed-up by developing a full quantum approach [71, 72]. This access can be achieved by oracularizing the environment's components, such as the transition probabilities and reward function. Other methods, based on the gradient estimation algorithm from [11] and developed in [73, 74], use quantum environments to directly compute the policy gradient as an output of a quantum procedure.

When trying to apply the standard quantum reinforcement learning techniques described above to Deep Hedging, one faces some problems that need to be resolved. One issue is that most quantum neural network models for policies have only been applied to discrete action spaces, while Deep Hedging has a continuous action space with constraints. Additionally, current algorithms for training quantum policies or value functions rely on solving the discounted Bellman equation, which is not suitable for hedging as the goal is defined by a utility function and the value function no longer follows the Bellman equation. Furthermore, building a quantum-accessible environment and approximating the policy gradient with quantum methods also poses a challenge, as these methods require finite action spaces and use amplitude estimation to approximate the value function and its gradient with respect to the policy.

Therefore, we aim to design a quantum reinforcement learning framework for Quantum Deep Hedging by addressing the challenges of standard quantum reinforcement learning techniques. In the subsequent subsections, we will outline the two methods we developed to overcome these challenges:

- **Using orthogonal layers:** We explore the application of quantum reinforcement learning methods to classical Deep Hedging environments using quantum orthogonal neural network architectures. This is in contrast to prior work that used variational circuits to represent parametrized quantum policies and value functions. To compare these quantum neural network architectures, we implemented policy-search Deep Hedging to train quantum policies and solve the classical environment. By using orthogonal layers, we aim to design a straightforward and effective method for enhancing Deep Hedging with quantum computing.

- **Using compound layers:** We propose a quantum native approach to the Deep Hedging problem, where we formulate Quantum Deep Hedging as a fully quantum reinforcement learning problem and solve it using actor-critic methods. Following the steps in [1, 2], we construct a quantum environment for Deep Hedging by providing quantum representations of the environment quantities and the trading goal. We then design specific quantum neural networks using compound layers and quantum reinforcement learning algorithms to solve the problem. Our approach is inspired by distributional reinforcement learning and leverages the properties of the quantum environment to provide a model-based quantum-enhanced solution to Deep Hedging.

Algorithm 1 Policy-Search Deep Hedging with Orthogonal Neural Networks

input Policy QNN π.
hyperparameters Number of episodes per training step N.
Initialize policy QNN with parameters $\boldsymbol{\phi}$.
while True **do**
  for episode $i = 1$ **to** N **do**
    for time-step $t = 0$ **to** T **do**
      Compute action $a_t^i := \pi_t^{\boldsymbol{\phi}_t}(s_t^i)$.
      Take action a_t^i and receive total reward $r_t^i := r_t(s_t^i, a_t^i)$.
    end for
    Compute total cumulative return $\widetilde{R}_0^i := \sum_{t=0}^{T} r_t^i$.
  end for
  Update policy parameters $\boldsymbol{\phi}$ with gradient descent to minimize

$$\widetilde{\mathcal{L}}(\boldsymbol{\phi}) := \frac{1}{\lambda} \log \frac{1}{N} \sum_{i=1}^{N} \exp\big(- \lambda \widetilde{R}_0^i \big).$$

end while
output Policy parameters $\boldsymbol{\phi}$.

5.1 Quantum Deep Hedging in Classical Environments

5.1.1 Classical Environment for Deep Hedging

We base our classical environment on the work in [1, 2]. To model the market state s_t, we assume that it can be represented by a sequence of market observations $\{M_t\}_{t=0}^{T}$ and provide a formal definition of the MDP for the environment described in Section 3.1. At each time-step t, the available market information M_t is described by n numerical quantities represented by a vector:

$$M_t \in \mathbb{R}^n.$$

This vector includes market information such as stock prices and other relevant financial data. In this setting, the market state s_t can be identified with the sequence of past and actual market observations $\{M_{t'}\}_{t'=0}^{t}$ up to time-step t:

$$s_t = (M_0, M_1, \ldots, M_t) \in \mathbb{R}^{n \times (t+1)}.$$

Furthermore, there are m available hedging instruments, such as stocks, options, or futures, that can be traded with high liquidity in the market. The classical Deep Hedging environment can be formally defined as a finite-horizon MDP as follows:

Definition 2 (Classical MDP for Deep Hedging)**.** *The classical MDP for Deep Hedging is a finite-horizon Markov Decision Process, defined by a tuple* $(\mathcal{S}, \mathcal{A}, p, r, T)$*. Here,* $\mathcal{S}$ *is the market state space, which can be decomposed into subsets* $\mathcal{S}_t \subset \mathbb{R}^{n \times (t+1)}$ *at each time-step* t*,* $\mathcal{A}$ *is the trading action space, which can also be decomposed into subsets* $\mathcal{A}_t \subset [0,1]^m$*,* p *is the transition model that can be represented by* $p_t : \mathcal{S}_t \to \Delta(\mathcal{S}_{t+1})$*,* r *is the reward function, which can be represented by* $r_t : \mathcal{S}_t \times \mathcal{A}_t \to \mathbb{R}$*, and* $T \in \mathbb{N}^*$ *is the time maturity of all hedging instruments.*

This formulation allows us to model the problem of Deep Hedging formally, where the objective is to choose a sequence of trading actions $\{a_t^\pi\}_{t=0}^{T}$ that optimizes the risk-adjusted

expected returns over the given time horizon T, given a sequence of market observations $\{M_t\}_{t=0}^T$. At every time-step, the policy π maps the current market state s_t to an action a_t^π, performing a sequence-to-sequence mapping from $\{M_t\}_{t=0}^T$ to $\{a_t^\pi\}_{t=0}^T$.

Building upon this classical MDP framework for Deep Hedging, we now introduce quantum reinforcement learning methods specifically designed for classical environments, leveraging orthogonal layers to enhance their performance and efficiency.

5.1.2 Quantum Reinforcement Learning methods for Classical Environments

Our first approach to Quantum Deep Hedging utilizes quantum orthogonal neural network architectures to parametrize the policy π. While in this part of our work we focus on parametrizing the policy, this approach could be extended to the value function as well.

The policy QNN $\pi(.;\boldsymbol{\phi})$ is a sequence-to-sequence model, which can be parametrized with $\boldsymbol{\phi} := \{\boldsymbol{\phi}_t\}_{t=0}^T$, one per time-step, that can be shared or not depending on the setting, such that $\pi_t^{\boldsymbol{\phi}_t}$ represents the neural network used at time-step t with parameters $\boldsymbol{\phi}_t$. The input time-series data $(M_0, M_1, \ldots, M_T)$ is preprocessed classically and transformed into a sequence of embeddings $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ in a high-dimensional feature space of dimension d. We use the quantum neural networks described in Section 4.3, which extract features from each $\boldsymbol{x}_t \in \mathbb{R}^d$ and may pass on information across time to produce the final output sequence $(\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_T)$. The number of hidden layers used in each neural network architecture is a hyper-parameter governed by factors like the complexity of the learning problem and the availability of resources. The output $\boldsymbol{y}_t \in \mathbb{R}^d$ can be further processed classically to obtain the desired result, which is the action $\pi_t^{\boldsymbol{\phi}_t}(s_t) \in \mathbb{R}^m$ in this case.

We can train these parametrized quantum policies using the policy-search Deep Hedging algorithm, introduced in [1, 2], by updating the set of parameters $\boldsymbol{\phi}$ using gradient descent to minimize the policy loss function $\mathcal{L}(\boldsymbol{\phi})$ defined as

$$\mathcal{L}(\boldsymbol{\phi}) := \frac{1}{\lambda} \log \mathbb{E}_{s_0}\left[\exp\Big(-\lambda \sum_{t=0}^{T} r_t\big(s_t, \pi_t^{\boldsymbol{\phi}_t}(\boldsymbol{s}_t)\big)\Big) \Big| s_0 \right].$$

The training procedure, outlined in Algorithm 1, proceeds as follows. At every iteration, it generates N trajectories $\{s_t^i\}_{t=0}^T$ before using the policy QNN to compute the sequence of actions. Using this sequence of actions, we can compute the cumulative return for each episode and then estimate the utility over these episodes to provide an estimate $\widetilde{\mathcal{L}}(\boldsymbol{\phi})$ of the policy loss function defined earlier, and then update the parameters $\boldsymbol{\phi}$.

The same principles of using quantum neural networks that use orthogonal layers within classical architectures can be extended to other deep reinforcement learning algorithms for Deep Hedging, such as actor-critic and value-based methods, as well as future approaches in Deep Hedging.

5.2 Quantum Deep Hedging in Quantum Environments

Here, we extend the classical Deep Hedging problem into the quantum case, starting by defining the quantum environment and the trading goal for Quantum Deep Hedging. Then, in order to develop quantum algorithms to solve this newly defined trading goal in the quantum environment, we connect to distributional reinforcement learning by showing that the value function in our environment can be expressed using a categorical distribution.

We will propose a model-based distributional approach to approximate this value function by constructing appropriate quantum unitaries and observables that approximate the value function and its distribution. Furthermore, we introduce two quantum algorithms:

Quantum Deep Hedging with expected actor-critic and Quantum Deep Hedging with distributional actor-critic, which are summarized in Algorithms 2 and 3, respectively. These algorithms use quantum neural network architectures with compound layers to approximate the policy and value functions.

5.2.1 Quantum Environment for Deep Hedging

We present here a method for converting the classical Deep Hedging problem into a quantum-native setup, which we refer to as the quantum environment for Deep Hedging. In order for this approach to be efficient, we make the state space finite, allowing the market states to be encoded into quantum states of the form $|s_t\rangle$. Moreover, at each time-step, the environment utilizes an oracle U_t^p to map the transition probabilities p_t to the amplitudes of a quantum state, which is a superposition of next states $|s_{t+1}\rangle$.

Note that the quantum environment introduced is still solving the classical Deep Hedging task and differs from previous work in quantum reinforcement learning. Previous work, such as [71–75], has employed quantum environments with finite action spaces to create model-based approaches. A key distinction in our approach is that our transition oracles do not necessitate encoding of the action a_t, as the Deep Hedging model assumes that actions have no impact on the transition probabilities, i.e., trading actions do not affect the market[37]. This is a valuable feature of our approach, as it removes the need for a quantum encoding of actions. Additionally, our approach does not require quantum access to the reward function through oracles. While previous work encodes the reward function into parts of a quantum state, for example in the amplitudes [71], in quantum registers [72, 73], or in the phases of the quantum states [74], it is unclear how to achieve this efficiently for the reward function associated with Deep Hedging and which is defined on a continuous action space. Consequently, we present an alternative formulation of quantum environments that incorporates the unique structure of the Deep Hedging MDP, as detailed in Definition 2.

To formally specify our quantum environment, we assume that the market information M_t at each time-step t can be represented as a n-bit binary string, which we encode in an n-qubit computational basis state $|M_t\rangle$. Note that $\langle M_t'|M_t\rangle = 0$ if $M_t' \neq M_t$. The quantum encoding $|s_t\rangle$ of the market state s_t can then be expressed as follows:

$$|s_t\rangle = |M_0\rangle \otimes |M_1\rangle \otimes \cdots \otimes |M_t\rangle \in \mathcal{H}^{\otimes n\times(t+1)}.$$

With the formalism described above, the oracle U_t^p encoding transitions probabilities as described by a classical transition function p_t can be written as follows:

$$U_t^p : |s_t\rangle \otimes |0\rangle^{\otimes n} \to |(\boldsymbol{s}_{t+1}|s_t)\rangle := \sum_{s_{t+1}} \sqrt{p_t(s_{t+1}|s_t)}|s_{t+1}\rangle.$$

We will now redefine the trading goal in the context of the quantum environment for Deep Hedging. While our focus is on the exponential utility E_λ as defined in Section 3.2, our approach can be applied to any risk measure that can be expressed as the expectation of a deterministic function over future returns. To evaluate the value function for a given policy π, we need to compute the expectation of the exponentiated rewards over future returns. Specifically, for a state s_t, the random variable $\exp(-\lambda R_t^\pi(\boldsymbol{s}_T))|s_t$, representing the exponentiated return, is a discrete distribution with values $\{R_t^\pi(s_T) \mid s_T \in \mathcal{S}_T\}$ and probabilities $p_t(s_T|s_t)$. We can express its expectation as the measurement of a quantum observable O_t^π in the quantum state $|(\boldsymbol{s}_T|s_t)\rangle$, where

$$O_t^\pi := \sum_{s_T} \exp(-\lambda R_t^\pi(s_T))|s_T\rangle\langle s_T|,$$

and $|(\boldsymbol{s}_T|s_t)\rangle$ is defined similarly to $|(\boldsymbol{s}_{t+1}|s_t)\rangle$ as

$$|(\boldsymbol{s}_T|s_t)\rangle := \sum_{s_T} \sqrt{p_t(s_T|s_t)}|s_T\rangle \in \mathcal{H}^{\otimes n\times(T+1)}.$$

This quantum state encodes the probabilities $p_t(s_T|s_t) := \mathbb{P}[s_T|s_t]$ in a superposition of all possible trajectories $(s_{t+1}, \ldots, s_T)$ of length $T-t$ and can be prepared by sequentially applying oracles $U_t^p, U_{t+1}^p, \ldots, U_{T-1}^p$ to $|s_t\rangle$ and $n \times (T-t)$ ancilla qubits. Denoting $\rho_t(\boldsymbol{s}_T|s_t) := |(\boldsymbol{s}_T|s_t)\rangle\langle(\boldsymbol{s}_T|s_t)|$, we have

$$\mathrm{Tr}[O_t^\pi \rho_t(\boldsymbol{s}_T|s_t)] = \sum_{s_T} p_t(s_T|s_t) \times \exp(-\lambda R_t^\pi(s_T)) = \mathbb{E}[\exp(-\lambda R_t^\pi(\boldsymbol{s}_T)|s_t].$$

Using this observable, we can redefine the trading goal in the quantum environment for Deep Hedging as finding an optimal policy π^* such that

$$\forall t,\ \forall s_t \in \mathcal{S}_t, \quad v_t^*(s_t) = -\frac{1}{\lambda}\log\left\{\inf_\pi \mathrm{Tr}\left[O_t^\pi \rho_t(\boldsymbol{s}_T|s_t)\right]\right\}.$$

Our next goal is to develop quantum algorithms to solve this newly defined trading goal in the quantum environment. In other words, our objective is to find the optimal policy π^* that minimizes the logarithm of the expectation of the quantum observable O_t^π, which represents the exponentiated rewards over future returns. To design our quantum algorithm, we now connect to distributional reinforcement learning by showing that the value function in our environment can be expressed using a categorical distribution.

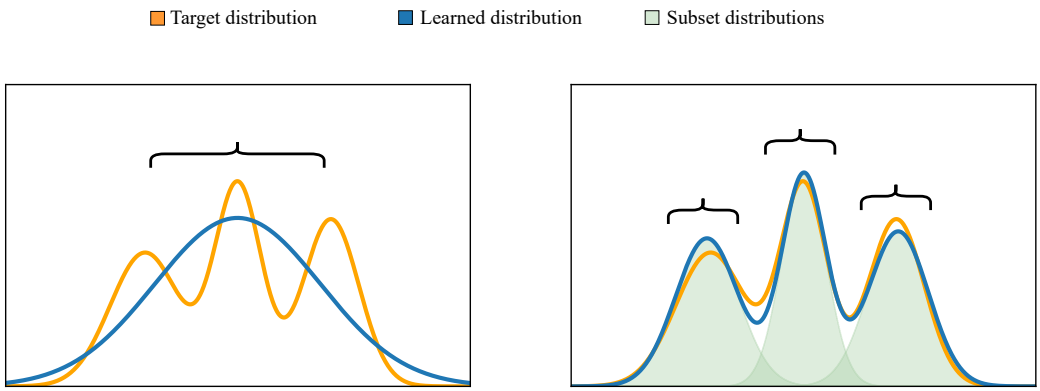
5.2.2 Connection with Distributional Reinforcement Learning

The connection between our proposed Quantum Deep Hedging approach and distributional reinforcement learning lies in the definition of value functions. In distributional reinforcement learning, the focus is on learning the probability distribution of the returns, as opposed to just the expected return value. This is done using neural networks [76, 77] that approximate the return distribution using categorical distributions. Similarly, in our Quantum Deep Hedging approach, the quantum observable O_t^π can be interpreted as a categorical distribution over all possible future returns, and our algorithms are designed to approximate this distribution and find the optimal policy π^* that minimizes the logarithm of the expectation of this distribution. Therefore, distributional reinforcement learning provides a useful framework for our approach.

In distributional reinforcement learning, value functions are defined using distributions, and categorical distributions are a common choice in many approximation schemes. A categorical distribution $\mathcal{P}^{\mathbf{z}}$, with a finite support $\mathbf{z} = \{z_1, z_2, \ldots, z_K\}$, is defined as a mixture of Dirac measures on each element of $\mathbf{z}$ and has the form $\mathcal{P}^{\mathbf{z}} := \sum_i p_i \delta_{z_i}$, where $p_i \geq 0$, $\sum_i p_i = 1$, and δ_{z_i} is the Dirac measure on z_i [26]. In other words, a categorical distribution is a probability distribution over a finite set of discrete outcomes. To formalize the notion of approximating distributions, we will use the Cramér distance (or ℓ_2 metric) defined as follows:

Definition 3 (Cramér distance)**.** *Given two distributions $\mathcal{P}, \mathcal{Q}$ over subsets of $\mathbb{R}$, with cumulative distribution functions (over $\mathbb{R}$) given by $F_\mathcal{P}, F_\mathcal{Q}$ respectively, the Cramér distance between the two distributions is defined as*

$$C_2(\mathcal{P}, \mathcal{Q}) = \left(\int_{\mathbb{R}} |F_\mathcal{P}(x) - F_\mathcal{Q}(x)|^2\, dx\right)^{1/2}.$$



(a) Learning over $\mathcal{S}_T$.

(b) Learning over multiple subsets of $\mathcal{S}_T$.

Figure 6: An example illustrating the process of learning expectations for each subset. The figures depict data generated from a trimodal distribution. In (6a), a single distribution is learned to match the expectation over the entire set of states. In (6b), the set is divided into three distinct subsets, with each peak representing the weighted and learned distribution within its corresponding subset. This improved approach provides a closer fit to the original data and effectively incorporates information about tails, offering a more accurate representation of the underlying distribution.

An important result from the distributional reinforcement learning literature [26, Proposition 1] shows the following properties of the Cramér distance: A categorical distribution $\mathcal{P}^{\mathbf{z}}$ over some support $\mathbf{z}$ can be projected onto a categorical distribution over a different support $\mathbf{z}'$ by a mapping Π_C (called the Cramér projection) that preserves the expectation ($\mathbb{E}[\mathcal{P}^{\mathbf{z}}] = \mathbb{E}[\Pi_C(\mathcal{P}^{\mathbf{z}})]$) while minimizing the Cramér distance between $\mathcal{P}^{\mathbf{z}}$ and $\Pi_C(\mathcal{P}^{\mathbf{z}})$, as long as the new support $\mathbf{z}'$ has a larger range, i.e $[\min \mathbf{z}, \max \mathbf{z}] \subset [\min \mathbf{z}', \max \mathbf{z}']$.

To establish a connection between our framework and distributional reinforcement learning, we can utilize the fact that sampling from a categorical distribution can be achieved by measuring a quantum observable. Specifically, the observable $O^{\mathbf{z}}$, defined as

$$O^{\mathbf{z}} := \sum_{b \in \{0,1\}^m} z_b |b\rangle\langle b|,$$

can be applied on the quantum state $|\mathbf{z}\rangle := \sum_b \sqrt{p_b}|b\rangle$ to sample from the categorical distribution $\mathcal{P}^{\mathbf{z}}$ with support $\mathbf{z}$. Here, the number of qubits m required to index all the outcomes is such that $K \leq 2^m$. Thus, measuring $O^{\mathbf{z}}$ in $|\mathbf{z}\rangle$ serves as a quantum representation of the distribution $\mathcal{P}^{\mathbf{z}}$ over the support $\mathbf{z}$.

In the Quantum Deep Hedging framework, the categorical distribution that models the returns distribution of a policy π at a time-step t can be represented by measuring O_t^π, the quantum observable defined in Section 5.2.1, in the quantum state $|(\boldsymbol{s}_T|s_t)\rangle$ that encodes the probabilities of the future trajectories given the current state s_t.

Constructing the quantum state $|(\boldsymbol{s}_T|s_t)\rangle$ requires quantum access to the environment, and it can be generated using the transition oracles $\{U_{t'}^p\}_{t'=t}^T$ and measured with the observable O_t^π. However, the observable O_t^π is not efficient to implement as its natural description requires classical computation of its eigenvalues and is dependent on the policy π, which changes during the training procedure. To address this, we propose a construction to approximately represent the distribution using a much simpler observable that is diagonal in the computational basis and has a spectrum independent of the specific reward structure.

This construction is detailed in the following proposition that demonstrates, using the Cramér projection, how an observable with a fixed support can approximate the value distribution while preserving its expectation.

Proposition 1. *Consider a support* $\mathbf{z}$ *of size* 2^m *such that, for any policy* π*, the following holds:*

$$\forall\, s_T \in \mathcal{S}_T, \quad \min_{b\in\{0,1\}^m} z_b \leq \exp(-\lambda R_t^\pi(s_T)) \leq \max_{b\in\{0,1\}^m} z_b,$$

where $m \geq 1$*. Then, there exists an observable* $O_t^{\mathbf{z}}$ *with eigenvalues in* $\mathbf{z}$ *that operates on* $n \times (t+1) + m$ *qubits and such that, for any deterministic policy* π*, there is a unitary* U_t^π *satisfying*

$$\forall s_t \in \mathcal{S}_t, \quad \mathrm{Tr}[O_t^{\mathbf{z}} \rho_t(\mathbf{z}|s_t)] = \mathrm{Tr}[O_t^\pi \rho_t(\boldsymbol{s}_T|s_t)],$$

where $|(\mathbf{z}|s_t)\rangle := U_t^\pi(|s_t\rangle \otimes |0\rangle^{\otimes m})$ *and* $\rho_t(\mathbf{z}|s_t) := |(\mathbf{z}|s_t)\rangle\langle(\mathbf{z}|s_t)|$*.*

Additionally, let $\mathcal{P}_t^\pi$ *and* $\mathcal{P}_t^{\mathbf{z}}$ *denote the distributions of the value function (outcome of measuring* O_t^π *in* $|(\boldsymbol{s}_T|s_t)\rangle$*) and the corresponding categorical projection onto the fixed support* $\mathbf{z}$ *(outcome of measuring* $O_t^{\mathbf{z}}$ *in* $|(\mathbf{z}|s_t)\rangle$*), respectively. If the cumulative distribution function of* $\mathcal{P}_t^\pi$ *is L-Lipschitz, then the Cramér distance between* $\mathcal{P}_t^\pi$ *and* $\mathcal{P}_t^{\mathbf{z}}$ *is such that:*

$$C_2(\mathcal{P}_t^\pi, \mathcal{P}_t^{\mathbf{z}}) \leq LZ^{3/2}/3 \cdot 2^m,$$

where $Z = (\max_{b\in\{0,1\}^m} z_b - \min_{b\in\{0,1\}^m} z_b)$*.*

Proof. Given a policy π and a state $s_t \in \mathcal{S}_t$, the distribution of the exponentiated returns is a categorical distribution, denoted as $\mathcal{P}_t^\pi$, from which we can obtain a categorical distribution over a support $\mathbf{z}$ using the Cramér projection. Specifically, we can project $\mathcal{P}_t^\pi$ onto the support $\mathbf{z}$ to obtain $\mathcal{P}_t^{\mathbf{z}} := \Pi_C(\mathcal{P}_t^\pi)$, which returns a $z_b \in \mathbf{z}$ with probability $p(z_b|s_t)$. We can then define a unitary that maps $|s_t\rangle|0\rangle^{\otimes m}$ to $|s_t\rangle|(\mathbf{z}|s_t)\rangle$, where $|(\mathbf{z}|s_t)\rangle := \sum_b \sqrt{p(z_b|s_t)}|b\rangle$ is a quantum state encoding the probabilities of $\mathcal{P}_t^{\mathbf{z}}$. The claim for a particular state s_t is satisfied when we measure this unitary with the observable

$$O_t^{\mathbf{z}} := I^{\otimes n\times(t+1)} \otimes \sum_{b\in\{0,1\}^n} z_b|b\rangle\langle b|,$$

where I is the identity operator acting on one qubit. Since the different quantum encodings $|s_t\rangle$ are orthogonal, a unitary U_t^π that performs the Cramér projection for all $s_t \in \mathcal{S}_t$ can be constructed. By known properties of the Cramér projection, the first requirement of matching expectations is satisfied.

We now consider the Cramér distance between the true and projected distributions. As the Cramér projection minimizes Cramér distance we obtain an upper bound by analyzing the distance from any projection onto the same support. Consider the projection assigns to each z_b the weight of the true distribution between z_b and z_{b-1} (where the subtraction is performed by viewing b as the binary representation of an integer). Let the true and projected cdfs be $F_{\mathcal{P}_t^\pi}$ and $F_{\mathcal{P}_t^{\mathbf{z}}}$ respectively. The square of the Cramér distance between these distributions is the sum of 2^m integrals of the form:

$$\int_{z_b}^{z_b+Z/2^m} (F_{\mathcal{P}_t^\pi}(x) - F_{\mathcal{P}_t^{\mathbf{z}}}(z_b))^2 dx \leq L^2Z^3/3 \cdot 2^{3m}.$$

Accumulating the 2^m terms and taking the square root, we have the necessary bound on the Cramér distance. □

Proposition 1 illustrates the use of quantum circuits to approximate the distributional value function by fixing a support $\mathbf{z}$, and measuring the observable $O_t^{\mathbf{z}}$ in the quantum states $|(\mathbf{z}|s_t)\rangle$ produced by the unitary U_t^π. We can hope to learn U_t^π by using existing classical distributional reinforcement learning algorithms for our quantum setting. However, this approach may be challenging for two reasons. First, most of the existing approaches in distributional reinforcement learning assume that the value function conforms to the discounted Bellman equation, which is not our case since we need to take into account the risk-adjusted measure. Second, the size of the quantum support increases exponentially with the number of qubits, making training impractical.

To overcome these challenges, we propose a new approach that utilizes the environment's model and exploits the structure and properties of our Hamming-weight preserving unitaries. This approach allows us to learn polynomial-sized distributions rather than exponential ones.

5.2.3 Distributional Value Function Approximation

We propose a model-based approach to learn distributional value functions in quantum environments. We introduce a new distributional reinforcement learning algorithm that differs from existing methods, in particular from [76] that fixes the support $\mathbf{z}$ and learns the probabilities for each element in the support and from [77] that fixes the probabilities (or quantiles) and learns the support. Our approach in fact splits the set of future trajectories into subsets and learns the expectation of the distribution within each subset. Because our approach is model-based, we can use the model to compute the probabilities of these subsets and calculate the overall expectation as well.

This approach can use any type of unitaries that preserve subspaces of some nature, while here we give a specific example using the Hamming-weight preserving quantum compound neural networks developed in Section 4.4. As we have seen, these compound neural networks preserve the Hamming-weight subspaces and thus allow us to split the set of future trajectories according to the Hamming-weight of their quantum representation.

In more detail, we partition the set of all complete trajectories $\mathcal{S}_T$ into $n(T-t)+1$ disjoint subsets $\mathcal{S}_T^{t,k}$, where $k = 0, \ldots, n(T-t)$, such that each subset contains terminal states with a Hamming-weight of k in their trajectories from $t+1$ to T. This allows us to decompose the superposition $|(\boldsymbol{s}_T|s_t)\rangle$ of all trajectories by grouping the future trajectories by Hamming-weight. Specifically, we express $|(\boldsymbol{s}_T|s_t)\rangle$ as a sum of terms corresponding to each subset, with each term given by $|(\boldsymbol{s}_T|s_t, k)\rangle$, where k is the Hamming-weight of the trajectories in the subset. We learn one expectation per subset, and by computing the probability of each subset, we can recover the overall expectation over all subsets. The probability that the future trajectory will have Hamming-weight k is denoted by $p_t(s_t, k) := \mathbb{P}[|\boldsymbol{s}_T| - |s_t| = k]$, and $|(\boldsymbol{s}_T|s_t, k)\rangle$ is the superposition of such trajectories defined as:

$$|(\boldsymbol{s}_T|s_t, k)\rangle := \frac{1}{\sqrt{p_t(s_t, k)}} \sum_{s_T \in \mathcal{S}_T^{t,k}} \sqrt{p_t(s_T|s_t)}|s_T\rangle.$$

In what follows, we will show that there exist Hamming-weight preserving unitaries that can approximate the expected value for each subset of future trajectories grouped by Hamming-weight, extending the result of Proposition 1.

Proposition 2. *Consider a support* $\mathbf{z}$ *of size* $2^{n(T-t)+2}$ *such that, for any policy* π *and for any Hamming-weight* $k = 0, \ldots, n(T-t)$*, the following holds:*

$$\forall\, s_T \in \mathcal{S}_T^{t,k}, \quad \min_{|b|=k+1} z_b \leq \exp(-\lambda R_t^\pi(s_T)) \leq \max_{|b|=k+1} z_b$$

Then, there exists an observable $O_t^{\mathbf{z}}$ with eigenvalues in $\mathbf{z}$ that operates on $n \times (T+1)+2$ qubits and such that, for any deterministic policy π, there is a Hamming-weight preserving unitary U_t^π satisfying:

$$\forall k,\ \forall s_t \in \mathcal{S}_t, \quad \mathrm{Tr}[O_t^{\mathbf{z}} \rho_t(\mathbf{z}|s_t, k+1)] = \mathrm{Tr}[O_t^\pi \rho_t(\boldsymbol{s}_T|s_t, k)]$$

where $|(\mathbf{z}|s_t, k{+}1)\rangle := U_t^\pi(|(\boldsymbol{s}_T|s_t, k)\rangle \otimes |01\rangle)$ *and* $\rho_t(\mathbf{z}|s_t, k{+}1) := |(\mathbf{z}|s_t, k{+}1)\rangle\langle(\mathbf{z}|s_t, k{+}1)|$.

Proof. Given a policy π and a state $s_t \in \mathcal{S}_t$, the distribution of the exponentiated returns restricted to future paths with Hamming-weight k is also a categorical distribution, denoted as $(\mathcal{P}_t^\pi)^{(k)}$, from which we can obtain a categorical distribution $(\mathcal{P}_t^{\mathbf{z}})^{(k)}$ over a support $\mathbf{z}^{(k)} := \{z_b \,|\, |b| = k+1\}$ using the Cramér projection. We can construct a Hamming-weight preserving unitary that maps $|(\boldsymbol{s}_T|s_t, k)\rangle \otimes |01\rangle$ to $|s_t\rangle \otimes |(\mathbf{z}^{(k)}|s_t)\rangle$, where: $|(\mathbf{z}^{(k)}|s_t)\rangle$ encodes the probabilities of $(\mathcal{P}_t^{\mathbf{z}})^{(k)}$. Measuring this state with the observable $O_t^{\mathbf{z}}$ as defined in the proof of Proposition 1 satisfies the claim for a specific state s_t and Hamming-weight k. Since the quantum states $|(\mathbf{z}^{(k)}|s_t)\rangle$ have different Hamming-weights, it is possible to construct a unitary operation that performs this mapping for all Hamming-weights. Similarly, as in Proosition 1, we can construct a unitary that performs this mapping for all states $s_t \in \mathcal{S}_t$. □

Regarding the above proof, there are two points to note. First, in order to calculate expectations on different subsets of future trajectories grouped by Hamming-weight, we added two ancilla qubits to satisfy the requirement of the Cramér projection for a support of size at least 2. This requirement is not satisfied for trajectories of Hamming-weight 0 or $n(T-t)$ without the addition of these ancilla qubits. As a result, the Hamming-weight of the measured quantum states has been shifted by +1. Second, if the subsets of trajectories with Hamming-weights 0 and $n(T-t)$ are empty, then only $n(T-t)$ qubits are needed instead of $n(T-t)+2$.

If we have Hamming-weight preserving unitaries, as described in Proposition 2, that produce the correct expectation on every subset, then we can obtain the overall expectation of the distributional value function. By loading the superposition over all future states $|(\boldsymbol{s}_T|s_t)\rangle$ using the transition oracles $\{U_{t'}^p\}_{t'=t}^{T-1}$ and applying the unitary U_t^π from Proposition 2, we can calculate directly the overall expectation without having to reconstruct the expectations per subspace and compute the overall expectation classically. Hamming-weight preserving unitaries act only inside the subspace to map each $|(\boldsymbol{s}_T|s_t, k)\rangle \otimes |01\rangle$ to $|(\mathbf{z}|s_t, k+1)\rangle$. Therefore, the overall expectation matches the expectation of the value function when we apply U_t^π to $|(\boldsymbol{s}_T|s_t)\rangle \otimes |01\rangle$, resulting in the state $|(\mathbf{z}|s_t)\rangle$ with density $\rho_t^\pi(\mathbf{z}|s_t))$. In other words, we have

$$\begin{aligned}
\mathrm{Tr}[O_t^{\mathbf{z}} \rho_t^\pi(\mathbf{z}|s_t))] &= \sum_{k=0}^{n(T-t)} \mathrm{Tr}[P_{k+1} \rho_t^\pi(\mathbf{z}|s_t)] \times \mathrm{Tr}[O_t^\pi \rho_t(\mathbf{z}|s_t, k+1)] \\
&= \sum_{k=0}^{n(T-t)} \mathrm{Tr}[P_k \rho_t(\boldsymbol{s}_T|s_t)] \times \mathrm{Tr}[O_t^\pi \rho_t(\boldsymbol{s}_T|s_t, k)] \\
&= \mathrm{Tr}[O_t^\pi \rho_t(\boldsymbol{s}_T|s_t)].
\end{aligned}$$

We have shown that for any policy π, a Hamming-weight preserving unitary exists at each time-step t, which can predict the expectation of the exponentiated returns on every subset of future paths accurately, as well as the overall expectation over all future paths by projecting the output states onto an observable $O_t^{\mathbf{z}}$ independent of π. We will now use quantum compound neural networks to parametrize these unitaries and provide

reinforcement learning algorithms to learn the expectation on every subset before using the overall expectation to improve the policy π.

5.2.4 Compound Neural Networks for Deep Hedging

In this section, we will discuss a general approach to constructing quantum neural networks using Hamming-weight preserving unitaries that can be used in Quantum Deep Hedging for quantum environments. We will explain how these networks can be used for both the policy and value function before providing algorithms for training them in Section 5.2.5. At each time-step, we design a quantum neural network that acts on $n(T+1)+2$ qubits. It takes the quantum encoding $|s_t\rangle$ as input on the first $n(t+1)$ qubits, applies the transition oracles on $n(T+1)$ qubits, and uses a compound neural network with two additional ancilla qubits (initialized to $|01\rangle$) to predict a value within some range $(o_{\min}, o_{\max})$. First, we will design the observable and then present the architecture of the quantum circuit for the value function, which can also be applied to the policy by adjusting the bounds accordingly.

To construct the observable, we need to define a support that covers the valid range of values $(o_{\min}, o_{\max})$ for every subset of Hamming-weights of size at least 2. For the value function, this range corresponds to the possible values of the exponentiated returns, which is crucial for the validity of the Cramér projection. For the policy, the support should only cover the range of valid actions. To design the support, we draw inspiration from [76] and define the support for each subset of Hamming-weight $k+1$ as having size $N_k = \binom{n(T-t)+2}{k+1}$ with uniformly spaced atoms. We can express this support as

$$O_t^{\mathbf{z}} = I^{\otimes n(t+1)} \otimes \sum_{k=0}^{n(T-t)} \sum_{|b|=k+1} \left(o_{\min} + (o_{\max} - o_{\min}) \frac{i_b}{N_k - 1}\right) |b\rangle\langle b|,$$

where i_b ranges from 0 to $N_k - 1$ and indexes all the quantum states with Hamming-weight $k+1$.

To construct the unitary, we begin with $|s_t\rangle \otimes |0\rangle^{\otimes n(T-t)} \otimes |01\rangle$ and apply the model to create a superposition over all future trajectories $|(\boldsymbol{s}_T|s_t)\rangle \otimes |01\rangle$. We can rewrite this superposition as the tensor decomposition $|s_t\rangle \otimes |\boldsymbol{M}_{t+1}, \ldots, \boldsymbol{M}_T\rangle \otimes |01\rangle$, since non-zero probability paths necesarily start from s_t. Next, we conditionally apply a compound layer to the last $n(T-t)+2$ qubits, controlled by the first $n(t+1)$ qubits. Instead of learning $|\mathcal{S}_t| = \mathcal{O}(2^{nt})$ parameters for each s_t, we control each of the first $n(t+1)$ qubits individually and learn a maximum number of $2n(t+1)$ parameters, two sets per control qubit.

Since the control qubits are always in a computational basis state, they can be removed and the appropriate unitary can be chosen classically for each past state. Thus, our compound neural network $U_t(\boldsymbol{\omega}_t)$ with parameters $\boldsymbol{\omega}_t$ is written as the product of $n(t+1)$ compound layers, each with parameters $\boldsymbol{\omega}_t^{i,0}$ or $\boldsymbol{\omega}_t^{i,1}$ depending on whether the i-th qubit is in state $|0\rangle$ or $|1\rangle$.

After applying the unitary $U_t(\boldsymbol{\omega}_t)$ as described earlier, we obtain the quantum state with density

$$\rho_t^{\boldsymbol{\omega}_t}(\mathbf{z}|s_t) = |(\mathbf{z}|s_t)\rangle\langle(\mathbf{z}|s_t)|,$$

where $|(\mathbf{z}|s_t)\rangle = U_t(\boldsymbol{\omega}_t)|(\boldsymbol{s}_T|s_t)\rangle \otimes |01\rangle$. Measuring the observable $O_t^{\mathbf{z}}$ in this state results in a value within the range $(o_{\min}, o_{\max})$. Furthermore, when using this circuit for the value function, we can apply $U_t(\boldsymbol{\omega}_t)$ to a specific Hamming-weight in order to predict a value only for that subset of trajectories.

In summary, we have described how to construct a quantum neural network for a given time-step t using the environment's model. These quantum neural networks will be

different for different environments and we provide a concrete example in Section 6.2 (see Figure 7). In the next section, we will use these networks to model the value function, adjust the same compound neural network such that it can be used for the policy and train the parameters of both networks using an actor-critic algorithm.

5.2.5 Quantum Reinforcement Learning Methods for Quantum Environments

We present an actor-critic approach to Quantum Deep Hedging, which is specifically designed for quantum environments. To represent the policy and the value function, we use compound neural networks, which were introduced in the previous section. For each time-step t, we use a compound neural network with $n(t+1)$ layers, where each layer is controlled by one of the first $n(t+1)$ qubits and acts on the remaining $n(T-t)+2$ qubits. We use the Brick architecture with logarithmic depth for each layer, resulting in an overall depth of $\mathcal{O}(nt \log n(T-t))$.

The value QNN v is parameterized by $\boldsymbol{\omega} := \{\boldsymbol{\omega}_t\}_{t=0}^{T}$, where $\boldsymbol{\omega}_t$ contains all the parameters used at time-step t. Each layer's parameters, $\boldsymbol{\omega}_t^i$, can be further split depending on the possible values of the i-th control qubit. We construct the observable $O_t^{\mathbf{z}}$ as defined in Section 5.2.4 on a support $\mathbf{z}$ that covers the bounds on the exponentiated return function. We assume knowledge of $\mathbf{z}$, which can be computed classically. In this case, the value QNN maps the state s_t to

$$v_t^{\boldsymbol{\omega}_t}(s_t) := -\frac{1}{\lambda} \log \mathrm{Tr}[O_t^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_t}(\mathbf{z}|s_t)].$$

Here, $\rho_t^{\boldsymbol{\omega}_t}(\mathbf{z}|s_t) := |(\mathbf{z}|s_t)\rangle\langle(\mathbf{z}|s_t)|$, and $|(\mathbf{z}|s_t)\rangle = U_t(\boldsymbol{\omega}_t)|(\boldsymbol{s}_T|s_t)\rangle \otimes |01\rangle$ is the output of the value QNN when applied to s_t. Similarly, we define the policy network π with parameters $\boldsymbol{\phi} := \{\boldsymbol{\phi}_t\}_{t=0}^{T}$. For each time-step, we define an observable $O_t^{\mathbf{a}}$ on a different support $\mathbf{a}$ that covers the range of valid actions (typically $[0,1]$) using our approach described in Section 5.2.4 such that

$$\pi_t^{\boldsymbol{\phi}_t}(s_t) := \mathrm{Tr}[O_t^{\mathbf{a}} \rho_t^{\boldsymbol{\phi}_t}(\mathbf{a}|s_t)],$$

where $\rho_t^{\boldsymbol{\phi}_t}(\mathbf{a}|s_t) := |(\mathbf{a}|s_t)\rangle\langle(\mathbf{a}|s_t)|$, and $|(\mathbf{a}|s_t)\rangle = U_t(\boldsymbol{\phi}_t)|(\boldsymbol{s}_T|s_t)\rangle \otimes |01\rangle$ is the output of the policy QNN when applied to s_t. If there is more than one hedging instrument, we can construct one policy QNN per instrument. However, this is not necessary for the value function since it evaluates the overall policy for all instruments.

To train the value network, we use two optimization objectives: the *distributional* and the *expected* losses. The distributional loss $\mathcal{L}_D$ takes the Hamming-weight of the current state into account when evaluating the expected reward, while the expected loss $\mathcal{L}_E$ only considers the expected reward at time t. Specifically, $\mathcal{L}_D(\boldsymbol{\omega})$ is defined as

$$\mathcal{L}_D(\boldsymbol{\omega}) := \mathbb{E}_{s_t,k}\Big[\big(\mathrm{Tr}[O_t^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_t}(\mathbf{z}|s_t,k+1))] - \exp\left(-\lambda R_t^{\pi}(\boldsymbol{s}_T)\right)\big)^2 | s_t, k\Big],$$

and $\mathcal{L}_E(\boldsymbol{\omega})$ is defined as

$$\mathcal{L}_E(\boldsymbol{\omega}) := \mathbb{E}_{s_t}\Big[\big(\mathrm{Tr}[O_t^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_t}(\mathbf{z}|s_t))] - \exp\left(-\lambda R_t^{\pi}(\boldsymbol{s}_T)\right)\big)^2\Big].$$

After updating the value parameters $\boldsymbol{\omega}$, we use them to build estimates of the value function and then update the policy parameters $\boldsymbol{\phi}$. Using the value estimates, we update the policy to minimize the loss, adapted from [37], defined as

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}) &:= \mathbb{E}_{s_t}\Big[\frac{1}{\lambda} \exp\big(-\lambda(r_t(s_t, \pi_t^{\boldsymbol{\phi}_t}(s_t)) + v_{t+1}^{\boldsymbol{\omega}_{t+1}}(\boldsymbol{s}_{t+1}))\big) | s_t\Big] \\ &:= \mathbb{E}_{s_t}\Big[\frac{1}{\lambda} \exp\big(-\lambda r_t(s_t, \pi_t^{\boldsymbol{\phi}_t}(s_t)) \times \mathrm{Tr}[O_{t+1}^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_{t+1}}(\mathbf{z}|\boldsymbol{s}_{t+1})]\big) | s_t\Big].\end{aligned}$$

Algorithm 2 Expected Actor-Critic Deep Hedging with Compound Neural Networks

input Policy QNN π, Value QNN v.
hyperparameters Number of episodes per training step N.
Initialize policy and value QNNs with parameters $\{\boldsymbol{\phi}_t\}_{t=0}^{T}, \{\boldsymbol{\omega}_t\}_{t=0}^{T}$.
while True **do**
  for episode $i = 1$ **to** N **do**
    for time-step $t = 0$ **to** T **do**
      Compute action $a_t^i := \mathrm{Tr}[O_t^{\mathbf{a}} \rho_t^{\boldsymbol{\phi}_t}(\mathbf{a}|s_t^i)]$.
      Take action a_t^i and receive total reward $r_t^i := r_t^+ - r_t^-$.
    end for
    for time-step $t = T$ **to** 0 **do**
      Compute total cumulative return: $\widetilde{R}_t^i = \sum_{t'=t}^{T} r_{t'}^i$.
    end for
  end for
  Update value parameters $\boldsymbol{\omega}$ with gradient descent to minimize:

$$\widetilde{\mathcal{L}}(\boldsymbol{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \left(\mathrm{Tr}[O_t^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_t}(\mathbf{z}|s_t^i))] - \exp(-\lambda \widetilde{R}_t^i) \right)^2.$$

  Update policy parameters $\boldsymbol{\phi}$ with gradient descent to minimize:

$$\widetilde{\mathcal{L}}(\boldsymbol{\phi}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \frac{1}{\lambda} \exp(-\lambda r_t^i) \times \mathrm{Tr}[O_{t+1}^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_{t+1}}(\mathbf{z}|s_{t+1}^i))]).$$

end while
output Policy parameters $\boldsymbol{\phi}$.

The training procedure for our approach is outlined in Algorithms 2 and 3. At each iteration, we generate N trajectories $\{s_t^i\}_{t=0}^{T}$ and use the policy QNN to compute the corresponding sequence of actions. Using this sequence of actions, we can compute the cumulative return for each episode and for each time-step and we use them to update the value network. When using the expected loss, we update the value parameters $\boldsymbol{\omega}$ such that we predict this cumulative return in expectation. In the distributional case, we need to compute the Hamming-weight of the future trajectory and update the parameters such that we predict the expectation for only that subspace. Once the value estimates are updated, we can use them to update the policy parameters $\boldsymbol{\phi}$.

5.2.6 Properties of Quantum Deep Hedging

Predicting the performance of any particular deep learning algorithm is difficult due to the complexity of the models used, and the often unpredictable behavior of the non-convex optimization that must be performed for training. It is however possible to examine some desirable global properties of the system that indicate (but do not guarantee) good performance. The structure of the presented Quantum Deep Hedging framework leads to some of these global properties, when specifically instantiated with Hamming-weight preserving unitaries (as in Section 5.2.3). Some of these properties have been hinted at throughout the text, and we summarize them here:

Algorithm 3 Distributional Actor-Critic Deep Hedging with Compound Neural Networks

input Policy QNN π, Value QNN v.
hyperparameters Number of episodes per training step N.
Initialize policy and value QNNs with parameters $\{\boldsymbol{\phi}_t\}_{t=0}^{T}, \{\boldsymbol{\omega}_t\}_{t=0}^{T}$.
while True **do**
  for episode $i = 1$ **to** N **do**
    for time-step $t = 0$ **to** T **do**
      Compute action $a_t^i := \mathrm{Tr}[O_t^{\mathbf{a}} \rho_t^{\boldsymbol{\phi}_t}(\mathbf{a}|s_t^i)]$.
      Take action a_t^i and receive total reward $r_t^i := r_t^+ - r_t^-$.
    end for
    for time-step $t = T$ **to** 0 **do**
      Compute total cumulative return: $\widetilde{R}_t^i = \sum_{t'=t}^{T} r_{t'}^i$.
      Compute Hamming-weight $k_t^i := |s_T^i| - |s_t^i|$.
    end for
  end for
  Update value parameters $\boldsymbol{\omega}$ with gradient descent to minimize

$$\widetilde{\mathcal{L}}(\boldsymbol{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \left(\mathrm{Tr}[O_t^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_t}(\mathbf{z}|s_t^i, k_t^i + 1))] - \exp(-\lambda \widetilde{R}_t^i) \right)^2.$$

  Update policy parameters $\boldsymbol{\phi}$ with gradient descent to minimize

$$\widetilde{\mathcal{L}}(\boldsymbol{\phi}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \frac{1}{\lambda} \exp(-\lambda r_t^i) \times \mathrm{Tr}[O_{t+1}^{\mathbf{z}} \rho_t^{\boldsymbol{\omega}_{t+1}}(\mathbf{z}|s_{t+1}^i)]).$$

end while
output Policy parameters $\boldsymbol{\phi}$.

- **Expressivity:** The central question for any learning algorithm is whether the parameterized models used are expressive enough to capture target models of interest. The "universality" of models such as deep neural networks has been a driving force in their adoption and utility. In our algorithms we do not use models that are universal in that they can express any quantum operation, however we show that they are expressive enough to capture the quantities of interest, which in our case is the true distribution of the value function. The primary challenge is that the distribution of the value function is on an unknown and potentially changing support. We show in Proposition 1 that our model that uses a fixed support and general parameterized unitaries on m-qubits can approximate the true distribution with error decaying exponentially in m. In Proposition 2, we specialize this result to the case where the value function distribution is constant on the Hamming-weight subspaces and we correspondingly use a fixed support with Hamming-weight preserving parameterized unitaries.

- **Generalization:** The number of possible futures in a deep-hedging environment grows exponentially with the time horizon T. In practice, our learning algorithm can only use a limited number of episodes (polynomial in T). We must therefore investigate the out-of-sample performance or generalization of our algorithm in this setting. This can however be guaranteed in our setting where we use the Quantum Compound Neural Networks. Our parameterized models consist of $\mathcal{O}(T)$ such networks, each

on $\mathcal{O}(T)$ qubits. From the definition in Section 4.4, each of the neural networks has $\mathcal{O}(T^2)$ parameters. As a consequence of results due to Caro and Datta [78], the pseudo-dimension of our parameterized model is polynomial in T. Therefore poly(T) episodes suffice to ensure that empirical risk minimization over our sample converges with high probability to the true optimal expressible model over the whole distribution of futures.

- **Trainability:** Finally we consider whether the task of optimizing the parameters for our model can be performed efficiently. The associated optimization problem is non-convex and thus training convergence cannot be guaranteed. We can show however that in our setting, the well-known "barren plateau" problem [21] does not arise. Each Quantum Compound Neural Network that we use is on $\mathcal{O}(T)$ qubits and has $\mathcal{O}(T)$ depth. Furthermore the loss function we measure can be constructed as a function of measurements in the computational basis (corresponding to a measuring the vector of observables Z on each qubit i). We initialize the parameters of the model as normal random variables with variance $\mathcal{O}(1/T)$. Theorem 2 ensures that the gradients decay only polynomially with the time horizon T.

6 Results

In the previous sections we introduced quantum methods for Deep Hedging, which use quantum orthogonal and compound neural networks within policy-search and actor-critic based reinforcement learning algorithms. In this section we present results of hardware experiments evaluating our methods on classical and quantum-accesible market environments.

We benchmarked our models for both classical and quantum environments using three different methods: simulating our quantum models on classical hardware assuming perfect quantum operations, simulating them on classical emulators that model the noise for quantum hardware, and applying our quantum models directly on the 20 qubit trapped-ion quantum processors Quantinuum H1-1, H1-2 [27]. Note that because orthogonal layers are efficiently simulatable classically, we can perform simulations for up to 64 qubits, while for the compound architectures that use the entire exponential space , we only simulated layers with up to 12 qubits.

In all experiments, the parameters of all the quantum compound neural networks were initialized using Gaussian initialization, and the training for all quantum neural networks was performed in exact classical simulation. In the following subsections, we give the details of the results.

6.1 Classical Market Environment

In the first part of our experiments, we consider Quantum Deep Hedging as described in Section 5.1 in classical market environments. We considered the environment from [1, 2] where the authors used Black-Scholes model to simulate the market state and evaluate hedging strategies. In this setup, the underlying asset is modeled using Geometric Brownian Motion (GBM), which is commonly used in finance to model stock prices.

A GBM is a continuous-time stochastic process $\boldsymbol{B}_t$ described by a stochastic differential equation

$$d\boldsymbol{B}_t = \mu \boldsymbol{B}_t \, dt + \sigma \boldsymbol{B}_t \, d\boldsymbol{W}_t,$$

where $\mu \in \mathbb{R}$ is the percentage drift and $\sigma \in \mathbb{R}^+$ is the percentage volatility. $\boldsymbol{W}_t$ corresponds to Brownian motion and thus $d\boldsymbol{W}_t \sim \mathcal{N}(0, dt)$.

Model	Utility		Number of parameters
	Without costs	With costs	
Feed-forward (Classical)	−2.868	−5.064	881
Feed-forward (Pyramid)	−2.873	−5.048	521
Feed-forward (Butterfly)	−2.874	−5.043	257
Recurrent (Classical)	−2.933	−5.075	881
Recurrent (Pyramid)	−2.939	−5.102	521
Recurrent (Butterfly)	−2.931	−4.854	257
LSTM (Classical)	−2.853	−4.743	569
LSTM (Pyramid)	−2.856	−4.755	457
LSTM (Butterfly)	−2.879	−4.787	217
Transformer (Classical)	−2.865	−4.713	1905
Transformer (Pyramid)	−2.876	−4.806	1305
Transformer (Butterfly)	−2.861	−4.822	865

Table 1: Comparison of expected utilities without and with transaction costs for models with classical and orthogonal layers using exact simulation over 256 paths and 30 trading days, including the number of trainable parameters.

For simulations, we assume one calendar year to be one unit of time increment and therefore set $dt = 1/252$ assuming 252 trading days in a calendar year. The market state s_t is represented by the sequence of past and actual market observations $\{M_{t'}\}_{t'=0}^{t}$, where $M_{t'}$ is the stock price at time-step t'. We used a European short call option with a strike price of $K = S_0$ as the instrument to be hedged. The time horizon was set to 30 trading days with daily rebalancing, and the percentage drift (μ) for the GBM was set to 0 and the percentage volatility (σ) was set to 0.2. Proportional transaction costs were utilized with a proportionality constant of 0.01. The training dataset comprised of 9.6×10^4 samples, whereas the testing dataset consisted of 2.4×10^4 samples. We compared Feed-forward, Recurrent, LSTM, and Transformer models, constructed using the framework described in Section 5.1. Here, the input sequence $(M_0, M_1, \ldots, M_T) \in \mathbb{R}^{(t+1)}$ corresponds to the mid-market price of the underlying equity. The outputs $a_t^\pi \in [0, 1]$ correspond to the model's delta for that time-step. The Feed-forward model is constructed using the *Feed Forward* architecture. Both Recurrent and LSTM models are built using the *Recurrent* architectures, where in the Recurrent model the hidden state passed onto the subsequent time-step is fixed to be the output of the previous time-step, i.e. the model's position on the hedging instrument at the previous time-step. The Transformer model we used in this work is constructed by adding the attention mechanism on top of the *Feed Forward* architecture.

Exact Simulations. To evaluate the behavior of quantum orthogonal neural networks, all four architectures (Feed-forward, Recurrent, LSTM, Transformer) were compared, both with classical linear and quantum orthogonal layers (with Pyramid and Butterfly circuits). A feature size of 16 was used for the linear layers in classical architectures, and 16 qubits were used for the orthogonal layers in quantum architectures. For Feed-forward and Recurrent models each hidden layer was repeated three times within the network. The LSTM model had one hidden cell constructed using four classical linear/quantum orthogonal layers. The Transformer model had three hidden layers followed by two classical lin-

Model	Utility		Number of circuits
	Simulator	Emulator	
Feed-forward (Butterfly)	−5.041	−5.155	960
Recurrent (Butterfly)	−5.006	−5.333	960
LSTM (Butterfly)	−4.809	−4.866	3840
Transformer (Butterfly)	−4.846	−5.176	2880

Table 2: Comparison of exact simulation and Quantinuum H1-1 emulator results for orthogonal layer models, evaluating expected utilities with transaction costs over 32 paths and 30 trading days, and showing the number of circuits emulated.

ear/quantum orthogonal layers for the attention mechanism. Parameters for all models were shared across time-steps. Noiseless classical simulations were performed for training and inference, with a batch of 256 paths. The results are presented in Table 1. We compared the achieved utilities with and without transaction costs and the number of training parameters. We observe that quantum orthogonal neural networks (Pyramid and Butterfly) achieve performance competitive with classical neural networks while using fewer trainable parameters and this holds for environments both with and without transaction costs. For the comparison, we have used the same classical and quantum architectures with the same layer sizes and we trained with identical hyperparameters. The quantum networks showed competitive performance and used fewer parameters due to the fact that every linear layer had been replaced with an orthogonal one. Let us also note that it might be possible to achieve parameter reduction with other classical methods (e.g. pruning). The Transformer and LSTM architectures demonstrated the highest model utilities among the studied architectures, while the quantum orthogonal Butterfly layers used fewest training parameters.

Hardware Emulations. To investigate the behavior of our quantum neural networks on current hardware, we employed Quantinuum H1-1 emulator [27] to perform inference on our models. We kept the same environment configuration and used a batch of 32 paths to perform inference. However, we downsized the network to a single layer as this allowed fewer circuit executions without significantly hampering model utilities. The Feed-forward and Recurrent architectures use one circuit evaluation per time-step, while the LSTM architectures use 4 circuit evaluations per time-step, and the Transformer architectures use 3 circuit evaluations per time-step. As the hardware architecture allowed for all-to-all connectivity, we used quantum orthogonal layers with a Butterfly circuit which enables log-depth circuits with linear number of two-qubit gates and thus are ideal for computations on near-term hardware. We used 1000 measurement shots per circuit evaluation to perform tomography over the unary basis and construct the output of each layer. The results are summarized in Table 2. The utility of the models is presented for two cases: when evaluated on a classical exact simulator and when evaluated on Quantinuum's hardware emulator. The table also summarizes the number of circuit evaluations needed to hedge 32 paths over 30 days with each model architecture. The results show that the LSTM architecture with Butterfly layers were most robust to noise as the model utilities on the simulator and emulator are relatively close. We also observed that the LSTM model achieved the highest utility on both cases.

Model	Utility	Terminal PnLs			
		Path 1	Path 2	Path 3	Path 4
LSTM (Classical)	−2.173	−2.578	−1.225	−1.420	−2.671
LSTM (Butterfly – Simulation)	−2.176	−2.586	−1.194	−1.439	−2.671
LSTM (Butterfly – Hardware)	−2.194	−2.610	−1.284	−1.488	−2.658
Transformer (Classical)	−2.167	−2.563	−1.219	−1.411	−2.673
Transformer (Butterfly – Simulation)	−2.195	−2.639	−1.242	−1.388	−2.672
Transformer (Butterfly – Hardware)	−2.539	−3.341	−1.355	−1.247	−2.713

Table 3: Comparison of exact simulation and Quantinuum H1-1 hardware results for orthogonal layer models, evaluating expected utilities and terminal PnLs with transaction costs over 4 paths and 5 trading days, evaluating the performance under hardware conditions.

Hardware Experiments. For our hardware experiments, we used the LSTM and Transformer models with 16-qubit Butterfly quantum circuits to perform inference on the Quantinuum H1-1 trapped-ion quantum processor [27]. We reduced the time horizon of the GBM to 5 days and considered models with transaction costs for a batch of 4 randomly chosen paths. We used the same model size as the ones used on hardware emulators which resulted in 80 circuit executions for the LSTM model and 60 circuit executions for the Transformer model. We present inference results for a model with a classical linear layer, and a quantum model with a butterfly quantum orthogonal layer simulated and executed on the quantum hardware. The results are presented in Table 3. In addition to model utilities, we also list the terminal Profit and Loss (PnL) for each path for a more fine-grained comparison. The results reveal that the LSTM architecture exhibits robustness to noise, consistent with the results obtained from the hardware emulator, as evidenced by the terminal PnL values of each path closely aligning with those of the simulations run on the hardware. Conversely, the Transformer's hardware execution demonstrates poorer performance compared to the simulation results.

6.2 Quantum Market Environment

In the second part of our experiments, we utilize quantum compound neural networks in various reinforcement learning algorithms in a quantum environment. Specifically, we implement the expected and distributional actor-critic algorithms, as described in Section 5.2, and compare them to the policy-search algorithm adapted for compound neural networks. To accomplish this, we first describe how to construct a quantum environment for Quantum Deep Hedging by adapting the classical environment used in Section 6.1. More specifically, we aim to build a quantum environment that mimics market dynamics following the Black-Scholes model, as described by a GBM.

To encode the dynamics of the Brownian motion in a quantum environment, we can use the fact that Brownian motions can be seen as the limit of a discrete random walk. Specifically, we can use a sequence of nT independent and identically distributed Bernoulli random variables $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_{nT}$ with mean $1/2$ to approximate $\boldsymbol{W}_T$, where T is the maturity and n is a hyperparameter that determines the precision of the approximation. Using this property, we can provide a discrete quantum environment for the Black-Scholes model, and approximate the price $\boldsymbol{B}_t$ at some time-step t by $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_{nt}$ as follows:

$$\boldsymbol{B}_t \approx B_0 \times \exp\Big((\mu - \frac{\sigma^2}{2})t + \frac{\sigma}{\sqrt{n}} \sum_{k=1}^{nt} (2\boldsymbol{b}_k - 1)\Big).$$

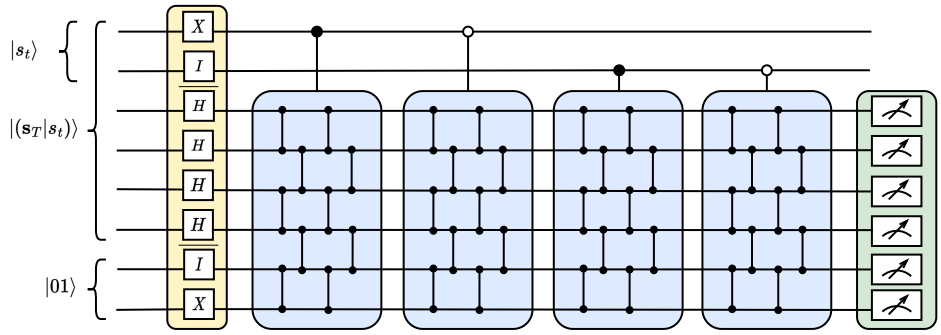


Figure 7: The quantum compound neural network using $\mathcal{O}(T)+2$ qubits for the Black-Scholes model. For loading the data (in yellow), first t qubits are used to encode past jumps of the market state. The next $\mathcal{O}(T-t)$ qubits encode the transition oracles which in the case of Black-Scholes corresponds to an equal superposition over all possible future jumps. The last two qubits are ancilla and are used to encode the state $|01\rangle$. Next for the unitaries (in blue), we used a architecture controlled on past market state. Based on the direction of jump, a different compound layer constructed using Brick architecture (Figure 2) is applied on $T-t+2$ qubits. Finally, each of the last $T-t+2$ qubits is measured (in green) independently. For $t=0$, the input state is an equal superposition of all possible future jumps followed by one unitary over $T+2$ qubits without control. As described in Section 5.2.4 the control qubits are always in a computational basis state. Thus, they can be removed for efficient hardware implementation, and the appropriate parameters for unitary acting on $T-t+2$ qubits can be chosen classically for each past state.

To obtain a sample B_t of $\boldsymbol{B}_t$, we sample nt Bernoulli variables $b_1,\dots,b_{nt}$, i.e., n Bernoulli variables per day. Thus, we define the encoding of the market observation M_t for a time-step $t>0$ as $|M_t\rangle := |b_{n(t-1)+1}\dots b_{nt}\rangle$, which contains all the jumps between t and $t+1$ and that can be encoded using n qubits. We define the encoding of the market state s_t at time-step t as the history of all previous jumps, i.e., $|s_t\rangle := |b_1 b_2 \dots b_{nt}\rangle$, from which we retrieve the price. Loading this quantum state can be done using a 1-depth circuit made with at most nt Pauli-X gates acting on nt qubits. Note that the number of qubits required to encode s_t here is nt and not $n(t+1)$ as in the general case, since the price at time-step 0 is fixed. For every time-step t, the transition model p_t can be oracularized by applying n Hadamard gates on an additional n qubits. The different transition oracles can be applied in parallel to build a superposition of all future trajectories and obtain

$$|(\boldsymbol{s}_T|s_t)\rangle = |s_t\rangle \otimes \sum_{b\in\{0,1\}^{n(T-t)}} \frac{1}{2^{n(T-t)/2}}|b\rangle.$$

In our experimental setup, we chose $n=1$ as the number of Bernoulli variables per day to approximate Brownian motion. We retained the same GBM parameters as in the classical environment ($\mu=0$ and $\sigma=0.2$), but set the instrument maturity to 10 days. Due to the limitations of simulating circuits with up to 12 qubits for compound architectures, we adjusted the time step increment to 30 trading days, instead of the usual 252. This means each time-step t should be changed to $t/30$ for the approximate GBM simulations, allowing us to capture short-term stock price fluctuations while maintaining the overall GBM behavior. The final payoff is a European short call option with a strike price of $k=1$. We also investigated cases with and without a transaction cost proportional to $\epsilon=0.002$.

Algorithm	Utility	
	Without costs	With costs
Policy-search	−4.064	−4.639
Expected actor-critic	−4.193	−4.668
Distributional actor-critic	−3.875	−4.424

Table 4: Comparison of compound neural networks trained using different algorithms with exact simulation, evaluating expected utilities over 16 paths and 10 trading days, both without and with transaction costs.

Algorithm	Utility		Number of
	Simulator	Emulator	circuits
Policy-search	−4.257	−4.277	160
Expected actor-critic	−4.528	−4.531	160
Distributional actor-critic	−4.185	−4.180	160

Table 5: Comparison of compound neural networks trained using different algorithms with simulation and Quantinuum H1-1 emulator results, evaluating expected utilities with transaction costs over 16 paths and 10 trading days.

We utilized the compound neural networks from Section 5.2.4 to represent the policy in the policy-search algorithm and both the policy and value in the actor-critic algorithms. We employed the Huber loss and scaled the value function to prevent exploding gradients. The algorithms were trained using classical simulations of the quantum circuits for 2000 steps with Adam optimizers, employing 3 random seeds and a batch of 16 generated episodes per training step. We selected the best parameters from these runs for a random selection of 16 paths and reported the inference results.

Exact Simulations. Here, the performance of the algorithms was evaluated through exact simulation on classical hardware using the Brick architecture with logarithmic depth per block for training the compound neural networks. The results, presented in Table 4, showed that policies trained using a distributional actor-critic algorithm yielded better utilities for this particular example. These results align with the findings by Lyle et al. [26], where minimizing a distributional loss led to better policies.

Hardware Emulations. We investigated the performance of the algorithms in presence of hardware noise by running inference on the Quantinuum H1-1 emulator [27] over 16 randomly chosen paths. To accommodate today's hardware limitations, the depth of circuits with large depth was reduced by using a fixed depth per block instead of logarithmic depth per block. We compared the inference results of the hardware emulator with exact simulation. Results presented in Table 5 show that quantum compound neural networks are noise-resilient, with similar utilities demonstrated between classical simulation and hardware emulation. Furthermore, our results show that the distributional policies outperformed the expected policies, and the expected policies outperformed the policies trained using the policy-search Deep Hedging algorithm.

Hardware Experiments. In the third part of our experiments, we performed inference on Quantinuum's H1-1 and H1-2 trapped-ion quantum processors [27] using policies trained

Algorithm	Utility	Terminal PnLs							
		Path 1	Path 2	Path 3	Path 4	Path 5	Path 6	Path 7	Path 8
Black Scholes	−4.884	−4.602	−5.373	−4.614	−4.263	−5.173	−5.030	−5.017	−4.962
Expected actor-critic (Simulation)	−3.547	+0.078	−6.204	−0.203	+0.967	−6.768	−3.071	−2.984	−6.689
Expected actor-critic (Hardware)	−3.501	+0.213	−6.666	−0.556	+1.067	−6.895	−2.315	−2.569	−6.556
Distributional actor-critic (Simulation)	−3.309	−1.807	−8.313	−3.803	+1.464	−2.736	−1.934	−2.669	−3.944
Distributional actor-critic (Hardware)	−3.369	−1.802	−8.214	−3.648	+1.367	−2.993	−2.047	−2.803	−4.200

Table 6: Comparison of compound neural networks trained using different algorithms with exact simulation and Quantinuum H1-1, H1-2 hardware results, evaluating expected utilities and terminal PnLs with transaction costs over 8 paths and 10 trading days, benchmarked against the standard Black-Scholes delta-hedging model.

via distributional and expected algorithms. We used a set of 8 randomly chosen paths and compared the terminal PnLs and utility in presence of transaction costs obtained via quantum hardware with exact classical simulations. The results are presented in Table 6. We also present the results from Black-Scholes delta hedge model for the setting . We note that the utility obtained from the hardware closely aligns with the emulation results, and the PnL values for the selected paths are also similar. Our study reveals that both the distributional and expected policies significantly outperformed the Black-Scholes delta hedge, with the distributional policy exhibiting the best overall performance.

7 Discussion

In this work we developed quantum reinforcement learning methods for Deep Hedging. These methods are based on novel quantum neural network architectures that utilize orthogonal and compound layers, and on a novel distributional actor-critic algorithm that takes advantage of the fact that quantum states and operations naturally deal with large distributions.

There are many potential advantages to using quantum methods to enhance the capabilities of Deep Hedging algorithms. First, for neural networks with deep architectures, as is the case for time-series data, feature orthogonality can improve interpretability, help to avoid vanishing gradients and result in faster and better training. Second, quantum compound neural networks can explore a larger dimensional optimization landscape and thus might train to more accurate models, once we ensure that barren plateau phenomena do not occur. Third, quantum neural networks are natively appropriate to be used in distributional reinforcement learning algorithms, which can lead to considerably better models, as we show for the toy example developed in this work. Finally, the quantum circuits that we need to implement in order to train competitive quantum models for Deep Hedging are rather small, since the number of qubits and depth of the quantum circuit is basically equal to the maturity time.

Note that our hardware experiments were done with a maturity of 10 days, due to the fact that we had to simulate the training of the quantum models on a classical computer which very soon becomes infeasible. In principle, one can train directly on the quantum computer, using for example a parameter shift rule to compute the gradients [50], in which case even with the current state of the quantum hardware (or with the hardware that will arrive in the next years) one can indeed train a Quantum Deep Hedging model for a maturity time of a month or more. In this case, the quantum model can no longer be simulated classically.

Moreover, we believe our quantum reinforcement learning methods have applications

beyond Deep Hedging, for example for algorithmic trading or option pricing, and it would be interesting to develop specific quantum methods for such problems. Note that in these use cases the training data can be produced efficiently, removing the bottleneck of loading large amounts of data onto the quantum computer.

The open questions regarding our work in quantum reinforcement learning are centered around three aspects. First, there is a need to expand the results regarding the trainability of the quantum neural networks proposed in this work to other settings. Second, the question of how to extend the quantum environment built for the GBM to other environments such as the Heston model studied in [37] arises. Finally, there is a need to design new distributional losses that make use of temporal difference methods to learn the value functions in the Deep Hedging context. One approach to this is using the theoretical framework that allows for such design as developed in [79], while another approach is using the moment matching approach as described in [80]. Currently, the work focuses on the expectation of the value function, but it is important to consider other moments that can be matched for both the overall expectation and the expectation per subspace.

Code Availability

The code for the numerical experiments in this paper is available upon request.

Acknowledgments

We would like to thank Phillip Murray from JPMorgan Chase for his help with the actor-critic experiments. Special thanks are also due to Tony Uttley, Jenni Strabley, and Brian Neyenhuis from Quantinuum for their assistance on the execution of the experiments on the Quantinuum H1-1, H1-2 trapped-ion quantum processors.
I.K. acknowledges support by projects EPIQ (ANR-22-PETQ-0007), QUDATA (ANR-18-CE47-0010), QOTP (QuantERA ERA-NET Cofund 2022-25), HPCQS (EuroHPC 2021-24).

Disclaimer

This paper was prepared for informational purposes with contributions from the Global Technology Applied Research center of JPMorgan Chase & Co. This paper is not a product of the Research Department of JPMorgan Chase & Co. or its affiliates. Neither JPMorgan Chase & Co. nor any of its affiliates makes any explicit or implied representation or warranty and none of them accept any liability in connection with this paper, including, without limitation, with respect to the completeness, accuracy, or reliability of the information contained herein and the potential legal, compliance, tax, or accounting effects thereof. This document is not intended as investment research or investment advice, or as a recommendation, offer, or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction.

References

[1] Hans Buehler, Lukas Gonon, Joseph Teichmann, and Ben Wood. "Deep hedging". Quantitative Finance **19**, 1271–1291 (2019). url: https://doi.org/10.1080/14697688.2019.1571683.

[2] Hans Buehler, Lukas Gonon, Josef Teichmann, Ben Wood, Baranidharan Mohan, and Jonathan Kochems. "Deep Hedging: Hedging Derivatives Under Generic Market Frictions Using Reinforcement Learning". SSRN Electronic Journal (2019). url: http://dx.doi.org/10.2139/ssrn.3355706.

[3] Shihao Gu, Bryan T. Kelly, and Dacheng Xiu. "Empirical Asset Pricing Via Machine Learning". SSRN Electronic Journal (2018). url: http://dx.doi.org/10.2139/ssrn.3159577.

[4] Hyeong Kyu Choi. "Stock Price Correlation Coefficient Prediction with ARIMA-LSTM Hybrid Model" (2018). url: https://doi.org/10.48550/arXiv.1808.01560.

[5] Yada Zhu, Giovanni Mariani, and Jianbo Li. "Pagan: Portfolio Analysis with Generative Adversarial Networks". SSRN Electronic Journal (2020). url: https://dx.doi.org/10.2139/ssrn.3755355.

[6] Kang Zhang, Guoqiang Zhong, Junyu Dong, Shengke Wang, and Yong Wang. "Stock Market Prediction Based on Generative Adversarial Network". Procedia Computer Science **147**, 400–406 (2019). url: https://doi.org/10.1016/j.procs.2019.01.256.

[7] Álvaro Cartea, Sebastian Jaimungal, and Leandro Sánchez-Betancourt. "Deep Reinforcement Learning for Algorithmic Trading". SSRN Electronic Journal (2021). url: https://dx.doi.org/10.2139/ssrn.3812473.

[8] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading". IEEE Transactions on Neural Networks and Learning Systems **28**, 653–664 (2017). url: https://doi.org/10.1109/TNNLS.2016.2522401.

[9] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. "A rigorous and robust quantum speed-up in supervised machine learning". Nature Physics 2021 17:9 **17**, 1013–1017 (2021). url: https://doi.org/10.1038/s41567-021-01287-z.

[10] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. "The Power of Block-Encoded Matrix Powers: Improved Regression Techniques via Faster Hamiltonian Simulation". In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019). Volume 132 of Leibniz International Proceedings in Informatics (LIPIcs), pages 33:1–33:14. Dagstuhl, Germany (2019). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. url: https://doi.org/10.4230/LIPIcs.ICALP.2019.33.

[11] András Gilyén, Srinivasan Arunachalam, and Nathan Wiebe. "Optimizing quantum optimization algorithms via faster quantum gradient computation". In Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Pages 1425–1444. (2019). url: https://doi.org/10.1137/1.9781611975482.87.

[12] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. "Variational quantum algorithms". Nature Reviews Physics **3**, 625–644 (2021). url: https://doi.org/10.1038/s42254-021-00348-9.

[13] Iordanis Kerenidis, Anupam Prakash, and Dániel Szilágyi. "Quantum Algorithms for Portfolio Optimization". In Proceedings of the 1st ACM Conference on Advances in

Financial Technologies. Pages 147–155. Zurich Switzerland (2019). ACM. url: https://doi.org/10.1145/3318041.3355465.

[14] Lucas Leclerc, Luis Ortiz-Guitierrez, Sebastian Grijalva, Boris Albrecht, Julia R. K. Cline, Vincent Elfving, Adrien Signoles, Loic Henriet, Gianni Del Bimbo, Usman Ayub Sheikh, Maitree Shah, Luc Andrea, Faysal Ishtiaq, Andoni Duarte, Samuel Mugel, Irene Caceres, Michel Kurek, Román Orús, Achraf Seddik, Oumaima Hammammi, Hacene Isselnane, and Didier M'tamon. "Financial Risk Management on a Neutral Atom Quantum Processor" (2022). url: https://doi.org/10.48550/arXiv.2212.03223.

[15] Dimitrios Emmanoulopoulos and Sofija Dimoska. "Quantum Machine Learning in Finance: Time Series Forecasting" (2022). url: https://doi.org/10.48550/arXiv.2202.00599.

[16] Patrick Rebentrost, Brajesh Gupt, and Thomas R. Bromley. "Quantum computational finance: Monte Carlo pricing of financial derivatives". Physical Review A **98**, 022321 (2018). url: https://doi.org/10.1103/PhysRevA.98.022321.

[17] João F. Doriguello, Alessandro Luongo, Jinge Bao, Patrick Rebentrost, and Miklos Santha. "Quantum Algorithm for Stochastic Optimal Stopping Problems with Applications in Finance". In François Le Gall and Tomoyuki Morimae, editors, 17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022). Volume 232 of Leibniz International Proceedings in Informatics (LIPIcs), pages 2:1–2:24. Dagstuhl, Germany (2022). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. url: https://doi.org/10.4230/LIPIcs.TQC.2022.2.

[18] Pradeep Niroula, Ruslan Shaydulin, Romina Yalovetzky, Pierre Minssen, Dylan Herman, Shaohan Hu, and Marco Pistoia. "Constrained quantum optimization for extractive summarization on a trapped-ion quantum computer". Scientific Reports**12** (2022). url: https://doi.org/10.1038/s41598-022-20853-w.

[19] Alexandre Ménard, Ivan Ostojic, Mark Patel, and Daniel Volz. "A game plan for quantum computing". McKinsey Quarterly (2020). url: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/a-game-plan-for-quantum-computing.

[20] Dylan Herman, Cody Googin, Xiaoyuan Liu, Alexey Galda, Ilya Safro, Yue Sun, Marco Pistoia, and Yuri Alexeev. "A survey of quantum computing for finance" (2022). url: https://doi.org/10.48550/arXiv.2201.02773.

[21] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. "Barren plateaus in quantum neural network training landscapes". Nature Communications **9**, 4812 (2018). url: https://doi.org/10.1038/s41467-018-07090-4.

[22] Iordanis Kerenidis, Jonas Landman, and Natansh Mathur. "Classical and Quantum Algorithms for Orthogonal Neural Networks" (2022). url: https://doi.org/10.48550/arXiv.2106.07198.

[23] Zebin Yang, Aijun Zhang, and Agus Sudjianto. "Enhancing Explainability of Neural Networks Through Architecture Constraints". IEEE Transactions on Neural Networks and Learning Systems **32**, 2610–2621 (2021). url: https://doi.org/10.1109/TNNLS.2020.3007259.

[24] Shuai Li, Kui Jia, Yuxin Wen, Tongliang Liu, and Dacheng Tao. "Orthogonal Deep Neural Networks". IEEE Transactions on Pattern Analysis and Machine Intelligence **43**, 1352–1368 (2021). url: https://doi.org/10.1109/TPAMI.2019.2948352.

[25] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz,

Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. "Discovering faster matrix multiplication algorithms with reinforcement learning". Nature **610**, 47–53 (2022). url: https://doi.org/10.1038/s41586-022-05172-4.

[26] Clare Lyle, Marc G. Bellemare, and Pablo Samuel Castro. "A Comparative Analysis of Expected and Distributional Reinforcement Learning". Proceedings of the AAAI Conference on Artificial Intelligence **33**, 4504–4511 (2019). url: https://doi.org/10.1609/aaai.v33i01.33014504.

[27] "Quantinuum H1-1, H1-2". https://www.quantinuum.com/ (2022). Accessed: November 15-22, 2022; December 7-12, 2022.

[28] Daniel J. Brod. "Efficient classical simulation of matchgate circuits with generalized inputs and measurements". Physical Review A**93** (2016). url: https://doi.org/10.1103/physreva.93.062332.

[29] Matthew L. Goh, Martin Larocca, Lukasz Cincio, M. Cerezo, and Frédéric Sauvage. "Lie-algebraic classical simulations for variational quantum computing" (2023). url: https://doi.org/10.48550/arXiv.2308.01432.

[30] Michał Oszmaniec, Ninnat Dangniam, Mauro E.S. Morales, and Zoltán Zimborás. "Fermion sampling: A robust quantum computational advantage scheme using fermionic linear optics and magic input states". PRX Quantum**3** (2022). url: https://doi.org/10.1103%2Fprxquantum.3.020328.

[31] Michael A. Nielsen and Isaac L. Chuang. "Quantum Computation and Quantum Information: 10th Anniversary Edition". Cambridge University Press. (2012). 1 edition. url: https://doi.org/10.1017/CBO9780511976667.

[32] R.S. Sutton and A.G. Barto. "Reinforcement Learning: An Introduction". IEEE Transactions on Neural Networks **9**, 1054–1054 (1998). url: https://doi.org/10.1109/TNN.1998.712192.

[33] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep Reinforcement Learning: A Brief Survey". IEEE Signal Processing Magazine **34**, 26–38 (2017). url: https://doi.org/10.1109/MSP.2017.2743240.

[34] Magnus Wiese, Lianjun Bai, Ben Wood, and Hans Buehler. "Deep Hedging: Learning to Simulate Equity Option Markets". SSRN Electronic Journal (2019). url: https://dx.doi.org/10.2139/ssrn.3470756.

[35] Hans Buehler, Phillip Murray, Mikko S. Pakkanen, and Ben Wood. "Deep Hedging: Learning to Remove the Drift under Trading Frictions with Minimal Equivalent Near-Martingale Measures" (2022). url: https://doi.org/10.48550/arXiv.2111.07844.

[36] Magnus Wiese, Ben Wood, Alexandre Pachoud, Ralf Korn, Hans Buehler, Murray Phillip, and Lianjun Bai. "Multi-Asset Spot and Option Market Simulation". SSRN Electronic Journal (2021). url: https://dx.doi.org/10.2139/ssrn.3980817.

[37] Phillip Murray, Ben Wood, Hans Buehler, Magnus Wiese, and Mikko Pakkanen. "Deep hedging: Continuous reinforcement learning for hedging of general portfolios across multiple risk aversions". In Proceedings of the Third ACM International Conference on AI in Finance. Page 361–368. ICAIF '22New York, NY, USA (2022). Association for Computing Machinery. url: https://doi.org/10.1145/3533271.3561731.

[38] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. "Quantum circuit learning". Physical Review A **98**, 032309 (2018). url: https://doi.org/10.1103/PhysRevA.98.032309.

[39] Dylan Herman, Rudy Raymond, Muyuan Li, Nicolas Robles, Antonio Mezzacapo, and Marco Pistoia. "Expressivity of Variational Quantum Machine Learning on the Boolean Cube" (2022). url: https://doi.org/10.1109/TQE.2023.3255206.

[40] Edward Farhi and Hartmut Neven. "Classification with Quantum Neural Networks on Near Term Processors". Technical report. Web of Open Science (2020). url: https://doi.org/10.48550/arXiv.1802.06002.

[41] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. "Data re-uploading for a universal quantum classifier". Quantum **4**, 226 (2020). url: https://doi.org/10.22331/q-2020-02-06-226.

[42] Jonas Landman, Natansh Mathur, Yun Yvonna Li, Martin Strahm, Skander Kazdaghli, Anupam Prakash, and Iordanis Kerenidis. "Quantum Methods for Neural Networks and Application to Medical Image Classification". Quantum **6**, 881 (2022). url: https://doi.org/10.22331/q-2022-12-22-881.

[43] Marcello Benedetti, Delfina Garcia-Pintos, Oscar Perdomo, Vicente Leyton-Ortega, Yunseong Nam, and Alejandro Perdomo-Ortiz. "A generative modeling approach for benchmarking and training shallow quantum circuits". npj Quantum Information **5**, 45 (2019). url: https://doi.org/10.1038/s41534-019-0157-8.

[44] Marcello Benedetti, Brian Coyle, Mattia Fiorentini, Michael Lubasch, and Matthias Rosenkranz. "Variational Inference with a Quantum Computer". Physical Review Applied **16**, 044057 (2021). url: https://doi.org/10.1103/PhysRevApplied.16.044057.

[45] Nico Meyer, Christian Ufrecht, Maniraman Periyasamy, Daniel D. Scherer, Axel Plinge, and Christopher Mutschler. "A Survey on Quantum Reinforcement Learning" (2022). url: https://doi.org/10.48550/arXiv.2211.03464.

[46] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. "Supervised learning with quantum-enhanced feature spaces". Nature **567**, 209–212 (2019). url: https://doi.org/10.1038/s41586-019-0980-2.

[47] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. "Effect of data encoding on the expressive power of variational quantum-machine-learning models". Physical Review A **103**, 032430 (2021). url: https://doi.org/10.1103/PhysRevA.103.032430.

[48] Francisco Javier Gil Vidal and Dirk Oliver Theis. "Input Redundancy for Parameterized Quantum Circuits". Frontiers in Physics **8**, 297 (2020). url: https://doi.org/10.3389/fphy.2020.00297.

[49] El Amine Cherrat, Iordanis Kerenidis, Natansh Mathur, Jonas Landman, Martin Strahm, and Yun Yvonna Li. "Quantum Vision Transformers" (2022). url: https://doi.org/10.48550/arXiv.2209.08167.

[50] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. "Evaluating analytic gradients on quantum hardware". Physical Review A **99**, 032331 (2019). url: https://doi.org/10.1103/PhysRevA.99.032331.

[51] Iordanis Kerenidis. "A method for loading classical data into quantum states for applications in machine learning and optimization". US Patent Application (2020). url: https://patents.google.com/patent/US20210319350A1.

[52] Sonika Johri, Shantanu Debnath, Avinash Mocherla, Alexandros Singk, Anupam Prakash, Jungsang Kim, and Iordanis Kerenidis. "Nearest centroid classification on a trapped ion quantum computer". npj Quantum Information **7**, 122 (2021). url: https://doi.org/10.1038/s41534-021-00456-5.

[53] Iordanis Kerenidis and Anupam Prakash. "Quantum machine learning with subspace states" (2022). url: https://doi.org/10.48550/arXiv.2202.00054.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett,

editors, Advances in Neural Information Processing Systems. Volume 30. Curran Associates, Inc. (2017). url: https://doi.org/10.48550/arXiv.1706.03762.

[55] Martin Larocca, Frédéric Sauvage, Faris M. Sbahi, Guillaume Verdon, Patrick J. Coles, and M. Cerezo. "Group-Invariant Quantum Machine Learning". PRX Quantum **3**, 030341 (2022). url: https://doi.org/10.1103/PRXQuantum.3.030341.

[56] Jiayao Zhang, Guangxu Zhu, Robert W. Heath Jr., and Kaibin Huang. "Grassmannian Learning: Embedding Geometry Awareness in Shallow and Deep Learning" (2018). url: https://doi.org/10.48550/arXiv.1808.02229.

[57] Xuchen You, Shouvanik Chakrabarti, and Xiaodi Wu. "A Convergence Theory for Over-parameterized Variational Quantum Eigensolvers" (2022). url: https://doi.org/10.48550/arXiv.2205.12481.

[58] Martin Larocca, Nathan Ju, Diego García-Martin, Patrick J. Coles, and M. Cerezo. "Theory of overparametrization in quantum neural networks" (2021). url: https://doi.org/10.1038/s43588-023-00467-6.

[59] Martin Larocca, Piotr Czarnik, Kunal Sharma, Gopikrishnan Muraleedharan, Patrick J. Coles, and Marco Cerezo. "Diagnosing Barren Plateaus with Tools from Quantum Optimal Control". Quantum **6**, 824 (2022). url: https://doi.org/10.22331/q-2022-09-29-824.

[60] Benoît Collins and Piotr Śniady. "Integration with Respect to the Haar Measure on Unitary, Orthogonal and Symplectic Group". Communications in Mathematical Physics **264**, 773–795 (2006). url: https://doi.org/10.1007/s00220-006-1554-3.

[61] Enrico Fontana, Dylan Herman, Shouvanik Chakrabarti, Niraj Kumar, Romina Yalovetzky, Jamie Heredge, Shree Hari Sureshbabu, and Marco Pistoia. "The Adjoint Is All You Need: Characterizing Barren Plateaus in Quantum Ansätze" (2023). url: https://doi.org/10.48550/arXiv.2309.07902.

[62] Michael Ragone, Bojko N. Bakalov, Frédéric Sauvage, Alexander F. Kemper, Carlos Ortiz Marrero, Martin Larocca, and M. Cerezo. "A Unified Theory of Barren Plateaus for Deep Parametrized Quantum Circuits" (2023). url: https://doi.org/10.48550/arXiv.2309.09342.

[63] Léo Monbroussou, Jonas Landman, Alex B. Grilo, Romain Kukla, and Elham Kashefi. "Trainability and Expressivity of Hamming-Weight Preserving Quantum Circuits for Machine Learning" (2023). url: https://doi.org/10.48550/arXiv.2309.15547.

[64] Kaining Zhang, Liu Liu, Min-Hsiu Hsieh, and Dacheng Tao. "Escaping from the Barren Plateau via Gaussian Initializations in Deep Variational Quantum Circuits" (2022). url: https://doi.org/10.48550/arXiv.2203.09376.

[65] Owen Lockwood and Mei Si. "Playing Atari with Hybrid Quantum-Classical Reinforcement Learning" (2021). url: https://doi.org/10.48550/arXiv.2107.04114.

[66] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. "Variational Quantum Circuits for Deep Reinforcement Learning". IEEE Access **8**, 141007–141024 (2020). url: https://doi.org/10.1109/ACCESS.2020.3010470.

[67] Owen Lockwood and Mei Si. "Reinforcement Learning with Quantum Variational Circuit". Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment **16**, 245–251 (2020). url: https://doi.org/10.1609/aiide.v16i1.7437.

[68] Yunseok Kwak, Won Joon Yun, Soyi Jung, Jong-Kook Kim, and Joongheon Kim. "Introduction to Quantum Reinforcement Learning: Theory and PennyLane-based Implementation". In 2021 International Conference on Information and Communica-

tion Technology Convergence (ICTC). Pages 416–420. Jeju Island, Korea, Republic of (2021). IEEE. url: https://doi.org/10.1109/ICTC52510.2021.9620885.

[69] Sofiene Jerbi, Casper Gyurik, Simon Marshall, Hans Briegel, and Vedran Dunjko. "Parametrized quantum policies for reinforcement learning". In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems. Volume 34, pages 28362–28375. Curran Associates, Inc. (2021). url: https://doi.org/10.48550/arXiv.2103.05577.

[70] Jen-Yueh Hsiao, Yuxuan Du, Wei-Yin Chiang, Min-Hsiu Hsieh, and Hsi-Sheng Goan. "Unentangled quantum reinforcement learning agents in the OpenAI Gym" (2022). url: https://doi.org/10.48550/arXiv.2203.14348.

[71] El Amine Cherrat, Iordanis Kerenidis, and Anupam Prakash. "Quantum reinforcement learning via policy iteration". Quantum Machine Intelligence **5**, 30 (2023). url: https://doi.org/10.1007/s42484-023-00116-1.

[72] Daochen Wang, Aarthi Sundaram, Robin Kothari, Ashish Kapoor, and Martin Roetteler. "Quantum algorithms for reinforcement learning with a generative model". In International Conference on Machine Learning. Pages 10916–10926. PMLR (2021). url: https://doi.org/10.48550/arXiv.2112.08451.

[73] Sofiene Jerbi, Arjan Cornelissen, Māris Ozols, and Vedran Dunjko. "Quantum policy gradient algorithms" (2022). url: https://doi.org/10.48550/arXiv.2212.09328.

[74] Arjan Cornelissen. "Quantum gradient estimation and its application to quantum reinforcement learning". Master Thesis (2018). url: http://resolver.tudelft.nl/uuid:26fe945f-f02e-4ef7-bdcb-0a2369eb867e.

[75] Hansheng Jiang, Zuo-Jun Max Shen, and Junyu Liu. "Quantum Computing Methods for Supply Chain Management". In 2022 IEEE/ACM 7th Symposium on Edge Computing (SEC). Pages 400–405. Seattle, WA, USA (2022). IEEE. url: https://doi.org/10.1109/SEC54971.2022.00059.

[76] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning". In Proceedings of the 34th International Conference on Machine Learning - Volume 70. Pages 449–458. ICML'17Sydney, NSW, Australia (2017). JMLR.org. url: https://doi.org/10.48550/arXiv.1707.06887.

[77] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. "Distributional Reinforcement Learning With Quantile Regression". Proceedings of the AAAI Conference on Artificial Intelligence**32** (2018). url: https://doi.org/10.1609/aaai.v32i1.11791.

[78] Matthias C. Caro and Ishaun Datta. "Pseudo-dimension of quantum circuits". Quantum Machine Intelligence**2** (2020). url: https://doi.org/10.1007%2Fs42484-020-00027-5.

[79] Hans Buehler, Murray Phillip, and Ben Wood. "Deep Bellman Hedging". SSRN Electronic Journal (2022). url: https://dx.doi.org/10.2139/ssrn.4151026.

[80] Thanh Nguyen-Tang, Sunil Gupta, and Svetha Venkatesh. "Distributional Reinforcement Learning via Moment Matching". Proceedings of the AAAI Conference on Artificial Intelligence **35**, 9144–9152 (2021). url: https://doi.org/10.1609/aaai.v35i10.17104.