

Quantum Quantile Mechanics: Solving Stochastic Differential Equations for Generating Time-Series

Annie E. Paine,* Vincent E. Elfving, and Oleksandr Kyriienko

A quantum algorithm is proposed for sampling from a solution of stochastic differential equations (SDEs). Using differentiable quantum circuits (DQCs) with a feature map encoding of latent variables, the quantile function is represented for an underlying probability distribution and samples extracted as DQC expectation values. Using quantile mechanics the system is propagated in time, thereby allowing for time-series generation. The method is tested by simulating the Ornstein-Uhlenbeck process and sampling at times different from the initial point, as required in financial analysis and dataset augmentation. Additionally, continuous quantum generative adversarial networks (qGANs) are analyzed, and the authors show that they represent quantile functions with a modified (reordered) shape that impedes their efficient time-propagation. The results shed light on the connection between quantum quantile mechanics (QQM) and qGANs for SDE-based distributions, and point the importance of differential constraints for model training, analogously with the recent success of physics informed neural networks.

derivatives and description of market dynamics. Potential applications of SDEs in finance lie in predicting stock prices, currency exchange rates and more.^[12]

A general system of stochastic differential equations can be written as^[11]

$$dX_t = f(X_t, t)dt + g(X_t, t)dW_t \quad (1)$$

where X_t is a vector of stochastic variables parameterized by time t (or other parameters). Deterministic functions f and g correspond to the drift and diffusion processes, respectively. W_t corresponds to the stochastic Wiener process. The stochastic component makes SDEs distinct from other types of partial differential equations, adding a non-differentiable contribution. This also makes SDEs generally difficult to treat. Several questions arise: how do we solve Equation (1), and what kind of

information do we want to get by solving SDEs? These are not trivial questions to answer, because one might be interested in different aspects of SDE-modeled processes.

First, the system of SDEs can be rewritten in the form of a partial differential equation for the underlying probability density function (PDF) $p(\mathbf{x}, t)$ of deterministic continuous variables, and then solved for the specified boundary conditions. The resulting equation is known as a Fokker–Planck (FP) or Kolmogorov equation.^[13] It can be exploited for calculating observables (averages) from $p(\mathbf{x}, t)$; in many cases these are the mean and the variance for the stochastic variables, $E[X_t] = \int \mathbf{x}p(\mathbf{x}, t)d\mathbf{x}$ and $\text{Var}[X_t] = E[X_t^2] - E[X_t]^2$, respectively. Here, a challenge arises when a well-defined boundary condition is missing, and instead some additional data is available. The task then falls into the category of data-driven machine learning problems, which has attracted attention recently.^[14,15] Also, a (potentially implicit) solution of the FP equation does not offer strategies to generate samples directly. Namely, drawing $\mathbf{x} \sim p(\mathbf{x}, t)$ from a complicated multidimensional distribution, at different t , represents another computationally expensive problem to solve.


Second, Equation (1) can be integrated using the Euler–Maruyama method,^[16] where the deterministic part of the differential equation (DE) is solved with stepwise propagation, and the Wiener process is modeled with a random number generator. This corresponds to the ensemble generation process (broadly speaking, *generative modeling*), which is computationally challenging for complex models. For multidimensional systems this is hindered by the curse of dimensionality. The task of solving random partial differential equations was recently addressed with

1. Introduction

Stochastic differential equations (SDEs) describe a broad range of phenomena. They emerge when dealing with Brownian motion and quantum noise.^[1] In physical sciences, SDEs are used for describing quantum dynamics,^[2] thermal effects,^[3] molecular dynamics,^[4] and they lie at the core of stochastic fluid dynamics.^[5,6] In biology SDEs help in the studies of population dynamics^[7] and epidemiology.^[8,9] They can also help to detect anomalies.^[10] SDEs are widely used in financial calculus,^[11] a fundamental component of all mechanisms of pricing financial

A. E. Paine, O. Kyriienko
Department of Physics and Astronomy
University of Exeter
Stocker Road, Exeter EX4 4QL, UK
E-mail: ap877@exeter.ac.uk

A. E. Paine, V. E. Elfving, O. Kyriienko
PASQAL SAS
2 av. Augustin Fresnel, Palaiseau 91120, France

 The ORCID identification number(s) for the author(s) of this article can be found under <https://doi.org/10.1002/qute.202300065>

© 2023 The Authors. Advanced Quantum Technologies published by Wiley-VCH GmbH. This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

DOI: 10.1002/qute.202300065

deep learning using physics-informed neural networks.^[17–19] Unsupervised generative modeling often requires using adversarial training^[20,21] and generative adversarial network (GAN) architecture.^[22] Moreover, in cases where the information about system parameters is limited, or exemplary datasets are generated from measurement and not known initial conditions, we arrive at the task of model discovery.^[23–27]

Quantum computers operate in a multidimensional state space and are intrinsically probabilistic. They offer computational advantage for sampling tasks.^[28–31] This was demonstrated experimentally with superconducting^[32] and optical^[33] quantum devices. Thus, quantum computing may be a prime candidate for disrupting the field of generative modeling. Yet, the challenge of performing the quantum generative modeling for practical tasks remains open. Ideally, discrete distributions can be loaded into a quantum register using an amplitude encoding $\sum_{k=1}^K u_k |k\rangle$, where amplitudes u_k contain information about the probability distribution and $|k\rangle$ are binary states for $\log(K)$ -qubit register. This allows for a quadratic speed-up when using the amplitude amplification protocol,^[34] and has numerous financial use-cases.^[35] However, representing arbitrary distributions in an amplitude-encoded form may be difficult (no scalable scheme exists yet), and the processing requires a large-scale fault-tolerant quantum computer.^[36] A strategy that is viable for near- and mid-term QC relies on variational quantum algorithms based on parametrized quantum circuits.^[37,38] These can be seen as quantum neural networks (QNNs).^[39–43] To date, several variational protocols for solving differential equations have been proposed for ordinary and partial differential equations,^[44–47] targeting near-term devices. For stochastic differential equations several algorithms have been explored that are based on wavefunction-based probability encoding^[48–52] and rely on complicated circuits. A strategy for near- and mid-term devices, SDE-based generative modeling can be addressed by data-based learning of probability distributions.

Motivated by classical GAN successes,^[22,53] various protocols for the adversarial training of generative quantum models were proposed, and coined as *qGANs*.^[54,55] Here, several sample generation strategies can be employed. One possible strategy relies on the variational wavefunction preparation as a distribution proxy, with the sample readout following the Born rule. This corresponds to a quantum circuit Born machine (QCBM) generator.^[56] In this case qGAN was applied to sampling from discrete distributions,^[57–63] preparing arbitrary quantum states,^[64,65] and being demonstrated experimentally with superconducting circuits for image generation.^[66] In this setting the sampling procedure is efficient, but its power depends on the register width and the generator is difficult to train at increasing scale. The latter comes from demanding requirements on the training set^[67] and barren plateaus for global cost functions.^[68] We also note that QCBM can be trained in other ways, including maximum mean discrepancy and various statistical divergences,^[56,69–71] with the goal of preparing distributions arising in financial applications. The second possible strategy relies on a QNN-based generator representing a *continuous qGAN*,^[67] recently demonstrated experimentally,^[72] where a feature map embedding^[73] is used for continuous latent variable distribution. In this case the sampling rate is reduced due to the expectation value measurement, but the model becomes trainable, and its power is ultimately limited

by the properties of feature maps and the adversarial training schedule (i.e., minimax game instead of loss minimization). The operation of qGANs was recently surveyed in Ref. [74]. Finally, other generator architectures are represented by quantum Boltzmann machines^[75–77] where sampling is based on thermal states of quantum Ising Hamiltonian, or recurrent neural networks for NMR quantum computers.^[78]

We note that the prior art in both classical and quantum generative modeling is either concentrated on representing probability density functions in model-driven approaches, or training the generator in a black-box fashion [e.g., (q)GANs] where no physical constraints are added. In this work, we suggest to shift the focus to the very tool that enables sampling—the *quantile function* (QF) (not to be mistaken with the quantile of a distribution). We propose to solve SDEs by rewriting them as differential equations for the quantile function and using their neural representation (classical or quantum neural networks). In the following, we use feature maps and differentiable quantum circuits (DQCs) to represent directly the quantile function of the probability distribution of the underlying SDE, and propagate them in time by solving the differential equations of quantile mechanics. This allows us to prepare a QNN-based generator that is trained from available data and yet is model-informed. While the expectation-based readout associated to the QNN structure requires multiple shots, the proposed approach has improved trainability compared to QCBM architecture and can work with sparse training data. Specifically, we benchmark the developed quantum quantile mechanics (QQM) approach^[79] using the Ornstein-Uhlenbeck model (a prototypical stochastic process being the base of many financial calculations). We show how to train QF from data and/or the known model at the initial point of time, and find a time-propagated QF that enables high-quality sampling. We then proceed to show that adversarial schemes (continuous GAN or qGAN) in fact train generators as a *reordered quantile function*. Analyzing the similarities and differences between the two methods, we find that quantile functions in their original meaning are suitable for time propagation, while reordered QFs have apparent difficulties for the task (see Discussion). Our work uncovers the possibilities for time series generation and *data-augmentation* enhanced by quantum resources.

2. Algorithm Section

First, the background is described for generative modeling from SDEs, and then we proceed to introduce the proposed QQM method.

2.1. Classical Sampling and Quantile Mechanics

Recall how a sample from a distribution can be obtained using a basic inversion sampling. In the following, a single stochastic process X_t (also written as X for brevity) is considered, and the generalization to multiple processes/dimensions is straightforward. First, a PDF $p(x)$ of a continuous variable $x \in \mathcal{X}$ being normalized over domain \mathcal{X} is taken. Next, a cumulative distribution function (CDF) is found out for the stochastic variable X defined as an integral $F_X(x) = \int_{-\infty}^x p(x') dx'$. This maps x to

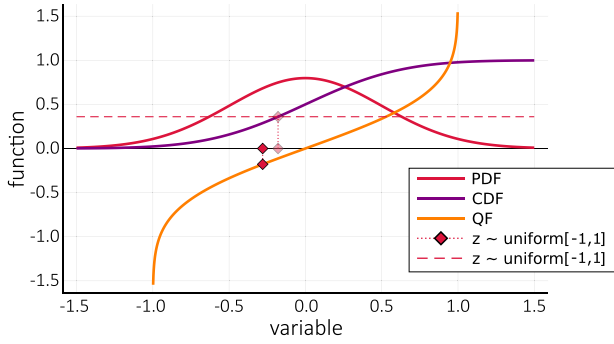


Figure 1. Illustrative example of sampling. We plot a normal probability density function as a red curve (PDF), being the Gaussian function with $(\mu, \sigma) = (0, 1/2)$. The domain is chosen as $\mathcal{X} = [-1.5, 1.5]$. The corresponding cumulative distribution function is presented by the purple solid curve (CDF). The quantile function for the corresponding CDF is shown in orange (QF). The red diamonds correspond to a randomly drawn latent variable z , and associated probability for the sample value, connected by dotted lines.

the cumulative probability value $p(X \leq x)$ lying in the $[0,1]$ range on the ordinate axis (see **Figure 1** for an illustration). Being a non-decreasing function, $F_X(x)$ has to be inverted to provide a sample. Generating a random number $z \in (-1, 1)$ from the uniform distribution, the equation $z = F_X(x)$ can be solved to find the corresponding sample. This requires finding the inverse of CDF, $F_X^{-1}(x)$. In most cases, this leads to a transcendental problem without a closed-form solution. This poses computational challenges and requires graphical solution methods.^[80] Finally, each random number $z \sim \text{uniform}(-1, 1)$ gives a random sample X from PDF of interest as $X = F_X^{-1}(z)$ (note that the range of z could be easily rescaled). The inverted CDF function is known as a *quantile function* of the continuous distribution, $F_X^{-1}(z) \equiv Q(z)$.

While generally quantile functions are difficult to get from PDFs, they can be obtained by solving nonlinear partial differential equations derived from SDEs of interest. This approach is called *quantile mechanics*. The quantile function $Q(z, t)$ can be obtained for any general SDE in the form (1) and its evolution is given by quantized FP as (see full derivation in Ref. [81])

$$\frac{\partial Q(z, t)}{\partial t} = f(Q, t) - \frac{1}{2} \frac{\partial g^2(Q, t)}{\partial Q} + \frac{g^2(Q, t)}{2} \left(\frac{\partial Q}{\partial z} \right)^{-2} \frac{\partial^2 Q}{\partial z^2} \quad (2)$$

where $f(Q, t)$ and $g(Q, t)$ are the drift and diffusion terms familiar from Equation (1). Equation (2) was solved as a function of latent variable z and time t . Once $Q(z, t)$ is known, by evaluating it at random uniform z 's as t progresses we can get full time series (trajectories) obeying the stochastic differential equation (1). In Appendix D, the case of solving reverse-time SDEs are also considered.

In general, quantile mechanics equations are not easy to solve. Power series solution as function approximation are known,^[81–84] as well as some simple examples.^[85] However, the difficulty arises when multidimensional problems are considered and generative modeling suffers from the curse of dimensionality. To harness

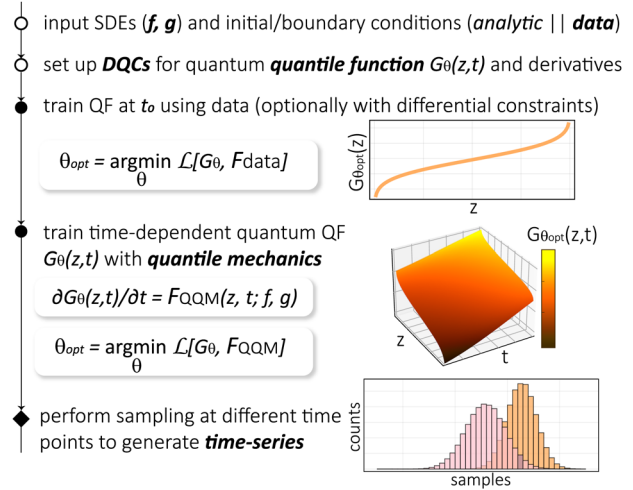


Figure 2. QQM workflow. Given the system of stochastic differential equations and initial data, differentiable quantum circuits are trained to represent the corresponding quantile function $G_\theta(z, t)$ as a function of a random latent variable (z) and propagate it in time (t) with quantum mechanics equations. The hybrid quantum-classical loop is used for optimizing variational parameters θ through loss function \mathcal{L} minimization based on data and differential equations for the initial and propagated QF. Evaluating $G_{\theta_{opt}}(z \sim \text{uniform}(-1, 1), t)$ at optimal angles, random values of the latent variable, and different time points t , we generate time series from SDE.

the full power of quantile mechanics, we thus propose to use neural representation of QFs. This is the first application of machine learning methods to use quantile-based sampling as per our knowledge, and it is envisaged that both classical and quantum ML could be used for the universal function approximation.^[73,86] Here, the use of quantum neural networks offers a potential to reproduce complex functions in high-dimensional space, including systems where strong correlations are important. In the following, quantum computing and quantile mechanics are combined to develop the *quantum quantile mechanics* approach. The sketch of the QQM workflow is shown in **Figure 2**. QFs are represented as quantum neural networks in the DQC form, thus exploiting the large expressivity of quantum-based learning. With this it is ensured that the ability to solve the problem is achieved. Then shown is how differential equations for quantile functions could be used for training differentiable quantum circuits. Second, the quantum quantile learning protocol is introduced for inferring QF from data and using QQM to propagate the system in time. This provides a robust protocol for time series generation and sampling. Finally, shown is that generative adversarial networks act as quantile functions for randomized association of the latent variable values and samples.

2.2. Quantum Quantile Mechanics

To represent a trainable (neural) quantile function, a parametrized quantum circuit is constructed using quantum embedding through feature maps $\hat{U}_\phi(x)$ (with ϕ labeling a mapping function),^[37,39,87] followed by variationally-adjustable circuit (ansatz) \hat{U}_θ parametrized by angles θ . The latter is routinely used in variational quantum algorithms,^[38] and for

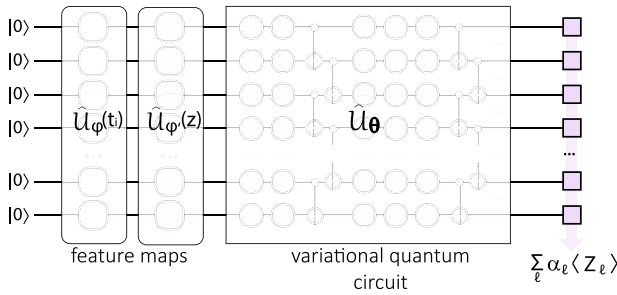


Figure 3. DQC layout. State preparation circuit for representing a time-dependent quantile function. Acting on the initial state, two feature maps, with \hat{R}_y using for time-dependence embedding and \hat{R}_x layer for the latent variable embedding. We use HEA for the variational search, and total Z magnetization as a cost function.

ML problems often has the hardware efficient ansatz (HEA) structure.^[88] The power of quantum feature maps comes from mapping $x \in \mathcal{X}$ from the data space to the quantum state $|\psi(x)\rangle = \hat{U}_{\phi}(x)|0\rangle$ living in the Hilbert space. We denote the initial state as $|0\rangle$, typically chosen as a product state $|0\rangle^{\otimes N}$. Importantly, the automatic differentiation of quantum feature maps allows derivatives to be represented as DQCs.^[45] The read-out is set as a sum of weighted expectation values.^[73] Following this strategy, a generator circuit $G(z, t)$ is assigned to represent a function parametrized by t (labels time as before), and the embedded latent variable z . The generator reads

$$G(z, t) = \langle 0 | \hat{U}_{\phi}(t)^{\dagger} \hat{U}_{\psi}(z)^{\dagger} \hat{U}_{\theta}^{\dagger} \left(\sum_{\ell=1}^L \alpha_{\ell} \hat{C}_{\ell} \right) \hat{U}_{\theta} \hat{U}_{\psi}(z) \hat{U}_{\phi}(t) | 0 \rangle \quad (3)$$

where $\hat{U}_{\phi}(z)$ and $\hat{U}_{\psi}(t)$ are quantum feature maps (possibly different), $\{\hat{C}_{\ell}\}_{\ell=1}^L$ represent L distinct Hermitian cost function operators, and $\{\alpha_{\ell}\}_{\ell=1}^L$ and θ are real coefficients that may be adjusted variationally. The overall cost function may be global, where projector on some specific bitstring was chosen, or local as a sum of Pauli terms. The choice of $\{\alpha_{\ell}\}_{\ell=1}^L$ enables to bias for/against certain operators/qubits information—for example for lower/higher frequencies as in Ref. [89]. The circuit structure is shown in **Figure 3**. One could also work with multiple latent variables and thus multidimensional distributions.

Our next step is developing the training procedure for G [or specifically for the generator's operator $\hat{U}_G(z) = \hat{U}_{\theta} \hat{U}_{\psi}(z)$], such that it represents QF for an underlying data distribution. Namely, we require that the circuit maps from the latent variable $z \in [-1, 1]$ to a sample $G(z) = Q(z)$.

The training requires a dataset $\{X_{\text{data}}\}$ generated from a probability distribution for the system we want to study (or measured experimentally), which serves as an initial/boundary condition. Additionally, we know the underlying processes that describe the system and which serve as differential constraints. The problem is specified by SDEs $dX_t = f_{\xi}(X_t, t)dt + g_{\xi}(X_t, t)dW_t$, where one explicitly states that the drift and diffusion functions $f_{\xi}(X_t, t)$ and $g_{\xi}(X_t, t)$ are parametrized by the vector ξ (time-independent). Here, ξ contains the vector of parameters which identify the exact SDE from its class of SDEs and could be found via model discovery.^[90] For instance, for the Ornstein–Uhlenbeck model,

which we refer to later, has $\xi = (\mu, \nu, \sigma)$ corresponding to the mean, rate of mean reversion, and volatility, respectively. One could also consider the situation when the SDE parameters for generating data similar to $\{X_{\text{data}}\}$ are known only in the first approximation $\xi^{(0)}$. Then, they could be adjusted during the training to have the best convergence, as used in the model discovery approach.^[23,25] The loss is constructed as a sum of data-based and SDE-based contributions, $\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{SDE}}$. The first part $\mathcal{L}_{\text{data}}$ is designed such that the data points in $\{X_{\text{data}}\}$ are represented by the trained QF. For this, the data is binned appropriately and collected in ascending order, as expected for any quantile function. Then, quantum circuit learning (QCL) is used as a quantum non-linear regression method^[87] to learn the quantile function. Note that such approach represents a data-frugal strategy, where the need for training on *all* data points is alleviated.

The second loss term \mathcal{L}_{SDE} is designed such that the learnt quantile function obeys probability models associated to SDEs. Specifically, the generator $G(z, t)$ needs to satisfy the quantized Fokker–Planck Equation (2). The differential loss is introduced using the DQC approach,^[45] and reads

$$\mathcal{L}_{\text{SDE}} = \frac{1}{M} \sum_{z, t \in \mathcal{Z}, \mathcal{T}} \mathfrak{D} \left[\frac{\partial G_{\theta}}{\partial t}, F(z, t, f, g, \frac{\partial G_{\theta}}{\partial z}, \frac{\partial^2 G_{\theta}}{\partial z^2}) \right] \quad (4)$$

where $\mathfrak{D}[a, b]$ denotes the distance measure for two scalars, and the loss is estimated over the grid of points in sets $(\mathcal{Z}, \mathcal{T})$. Here, $M = \text{car}(\mathcal{Z})\text{car}(\mathcal{T})$ is the total number of points. The function $F(z, t, f, g, \partial_z G_{\theta}, \partial_{zz}^2 G_{\theta})$ denoting the RHS for Equation (2), or any other differential constraint, is also introduced. Using a differential equation based constraint is a core principle of physics-informed machine learning and provides different fitting behavior versus regression due to the requirements on derivative values as well as function values.

Other important ingredients of the QQM method include the calculation of second-order derivatives for the feature map encoded functions (as required was \mathcal{L}_{SDE}), and the proposed treatment of initial/boundary conditions for multivariate function. These technical details are described in Appendix A. Once the training is set up, the loss is minimized using a hybrid quantum-classical loop where optimal variational parameters θ_{opt} (and α_{opt}) are searched for using non-convex optimization methods.

This process could be scaled to higher dimensions of encoded features. The first needed step is to encode each of the independent variables. There are many ways how this could be achieved and an investigation of multivariate encoding is a topic of ongoing research. A simple way to introduce several variables is layering several feature maps one after another. This model could be differentiated with respect to each variable. However, the number of terms to measure grows linearly with the number of variables. One also needs to be aware of the required increase of the training grid size. This issue occurs in classical machine learning, and it was shown that classical neural networks could work well in multiple dimensions.^[91]

3. Results

In this section, we present numerical experiments for training DQCs to represent time-dependent quantile functions, as well as qGAN training.

3.1. QQM-Based Generative Modeling

In the next subsections, we present numerical simulations of generative modeling. In the first part, we apply the developed quantum quantile mechanics approach for solving a specific SDE, and demonstrate a data-enabled operation. In the second part, we cover the so-called quantum generative adversarial network (qGAN) that was previously used for continuous distributions, and show numerical results for solving the same problem. The two approaches are then compared in the next section (Discussion).

3.1.1. Ornstein–Uhlenbeck for Financial Forecasting and Trading

To validate the QQM approach and perform time series forecasting, we pick a prototypical test problem. As an example we choose the Ornstein–Uhlenbeck (OU) process. In financial analysis its generalization is known as the Vasicek model.^[92] It describes the evolution of interest rates and bond prices.^[93] This stochastic investment model is the time-independent drift version of the Hull–White model^[94] widely used for derivatives pricing. We also note that OU describes dynamics of currency exchange rates, and is commonly used in Forex pair trading—a primary example for quantum generative modeling explored to date.^[70,71] Thus, by benchmarking the generative power of QQM for OU we can compare it to other strategies (valid at fixed time point).

The Ornstein–Uhlenbeck process is described by an SDE with an instantaneous diffusion term and linear drift. For a single variable process X_t the OU SDE reads

$$dX_t = \nu(\mu - X_t)dt + \sigma dW_t \quad (5)$$

where the vector of underlying parameters $\xi = (\nu, \mu, \sigma)$ are the speed of reversion ν , the long-term mean level μ , and the degree of volatility σ . The corresponding Fokker–Planck equation for the probability density function $p(x, t)$ reads

$$\frac{\partial p(x, t)}{\partial t} = -\nu \frac{\partial}{\partial x} ((\mu - x)p) + \frac{\sigma^2}{2} \frac{\partial^2 p}{\partial x^2} \quad (6)$$

When rewritten in the quantized form, it becomes a PDE for the quantile mechanics,

$$\frac{\partial Q(z, t)}{\partial t} = \nu [\mu - Q(z, t)] + \frac{\sigma^2}{2} \left(\frac{\partial Q}{\partial z} \right)^{-2} \frac{\partial^2 Q}{\partial z^2} \quad (7)$$

which follows directly from the generic Equation (2). In the following, we take the speed of reversion to be positive, $\nu > 0$ and adjust the long-term mean level to zero, $\mu = 0$.

Having established the basics, we train the differentiable quantum circuit to match the OU QF. First, for the starting point of time, we train the circuit to represent a quantile function based on available data (see the workflow chart in Figure 2 and the discussion below). Next, having access to the quantum QF at the starting point, we evolve it in time solving the equation

$$\frac{\partial G(z, t)}{\partial t} = -\nu G(z, t) + \frac{\sigma^2}{2} \left(\frac{\partial G}{\partial z} \right)^{-2} \frac{\partial^2 G}{\partial z^2} \quad (8)$$

as required by QM [Equation (7)]. This is the second training stage in the workflow chart shown in Figure 2.

To check the results, we use the analytically derived PDF valid for the Dirac delta initial distribution $p(x, t_0) = \delta(x - x_0)$ peaked at x_0 that evolves as

$$p(x, t) = \sqrt{\frac{\nu}{\pi\sigma^2(1 - \exp[-2\nu(t - t_0)])}} \times \exp \left[-\frac{\nu(x - x_0 \exp[-\nu(t - t_0)])^2}{\sigma^2(1 - \exp[-2\nu(t - t_0)])} \right] \quad (9)$$

and we can write the OU QF evolution as

$$Q(z, t) = x_0 \exp[-\nu(t - t_0)] + \sqrt{\frac{\sigma^2}{\nu} (1 - \exp[-2\nu(t - t_0)])} \operatorname{inverf}(z) \quad (10)$$

where $\operatorname{inverf}(x)$ denotes the inverse error function.^[95] This provides us a convenient benchmark of a simple case application and allows assessing the solution quality. Additionally, we use Euler–Maruyama integration to compare results with the numerical sampling procedure with fixed number of shots.

3.1.2. Ornstein–Uhlenbeck with Analytic Initial Condition

To highlight the generative power of the QQM approach we start by simulating the OU evolution $G(z, t)$ starting from the known initial condition. This is set as $G(z, 0) = Q(z, 0)$ being the analytic solution [Equation (10)] or can be supplied as a list of known samples associated to latent variable values. To observe a significant change in the statistics and challenge the training, we choose the dimensionless SDE parameters as $\nu = 1$, $\sigma = 0.7$, $x_0 = 4$, and $t_0 = -0.2$ such that we evolve a narrow normal distribution with strongly shifted mean into a broad normal distribution at $\mu = 0$.

We use DQCs with $N = 6$ qubits and a single cost operator being the total Z magnetization, $\hat{C} = \sum_{j=1}^N \hat{Z}_j$. This cost

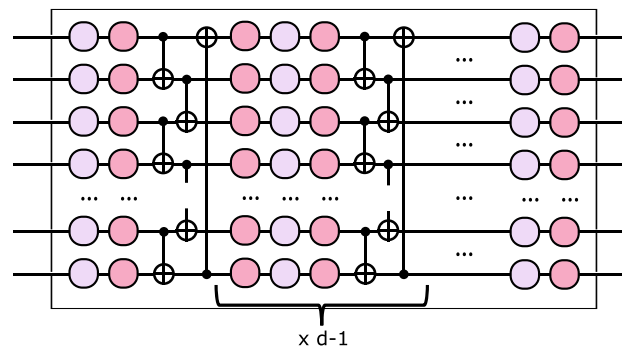


Figure 4. Circuit diagram of variational circuit used in simulations. Purple/pink boxes represent Pauli X/Y rotation gates. Entangling layer consists of CNOT gates on nearest neighbors and endpoints. Variational circuit of depth d is made up of initial layer of rotations $R_x R_z$ on each qubit followed by entangling layer composed of CNOTs. Then follows $d - 1$ implementations of rotation layer ($R_z R_x R_z$ on each qubit), then entangling layer. The circuit terminates after a final set of two rotations $R_z R_x$ on each qubit.

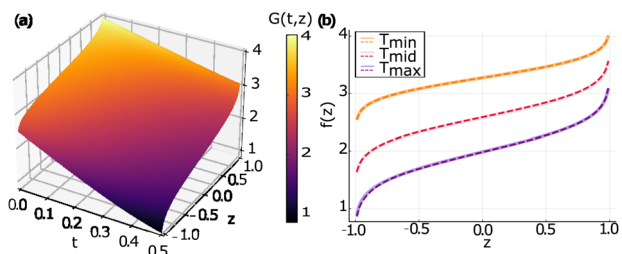


Figure 5. Training time evolution of Ornstein-Uhlenbeck process. The results are shown for runs with analytic initial condition and parameters chosen as $\nu = 1, \sigma = 0.7, x_0 = 4$. a) Surface plot for the trained DQC-based quantile function $G(z, t)$ that changes in time. b) Slices of the quantum quantile function $G(z, t)$ shown at discrete time points $t = 0$ (labeled as T_{\min} hereafter), $t = 0.25$ (T_{mid}), and $t = 0.5$ (T_{\max}).

operator was chosen as it collects information from all qubits and it is local (here, the sum of single-qubit operators). Local cost operators were shown to have favorable behavior as compared to global when considering barren plateaus.^[68] To build a model that does not bias certain qubits, we choose to have equal coefficients for the operator on each qubit. For simplicity, we train the circuit using a uniformly discretized grid with \mathcal{Z} containing 21 points from -1 to 1 , and \mathcal{T} containing 20 values from 0.0 to 0.5 , totalling 420 points in the full grid. To encode the function, we use the product-type feature maps^[45,87] chosen as $\widehat{U}_\phi(t) = \bigotimes_{j=1}^N \exp[-i \arcsin(t) \widehat{Y}_j/2]$ and $\widehat{U}_\phi(z) = \bigotimes_{j=1}^N \exp[-i \arcsin(z) \widehat{X}_j/2]$. The variational circuit corresponds to HEA with the depth of six layers of generic single-qubit rotations plus nearest-neighbor CNOTs, the circuit diagram of ansatz from is shown in **Figure 4**. We exploit the floating boundary handling,^[45] and choose a mean squared error (MSE) as the distance measure, $\mathfrak{D}[a, b] = (a - b)^2$ of our loss in Equation (4). Due to floating boundary handling being used, there is no $\mathcal{L}_{\text{data}}$ term in the loss function. The system is optimized for a fixed number of epochs, being 750, and we use the Adam optimizer for gradient-based training of variational parameters θ . We implement this workflow with a full quantum state simulator in a noiseless setting. This, as well as the other simulations presented, is realized in Yao.jl^[96]—a Julia package that offers state-of-the-art performance. For measurement infinite shots are used. Gradients calculations are simulated with an in-built function of Yao.jl which utilizes automatic differentiation techniques (back propagation) or with application of the parameter shift rule. Results for a limited number of shots are shown in Appendix G.

We present the results of DQC training in **Figure 5**. In **Figure 5a**, we show the trained quantum QF as a function of time t and the latent variable z . Choosing three characteristic points of time $t = \{0.0, 0.25, 0.50\}$ that we label as $\{T_{\min}, T_{\text{mid}}, T_{\max}\}$, we plot the corresponding quantile functions at these times (**Figure 5b**). The dashed curves from the DQC training closely follow ideal QFs shown by solid curves. Additionally, we note we observed the training loss smoothly and rapidly converging in 250 epochs as the circuit is expressive enough to represent changes of initial QF at increasing time, and thus providing us with evolved $G(z, t)$.

Next, we perform sampling and compare the histograms coming from the Euler–Maruyama integration of OU SDE^[16,97] and the QQM training presented above. The results are shown in

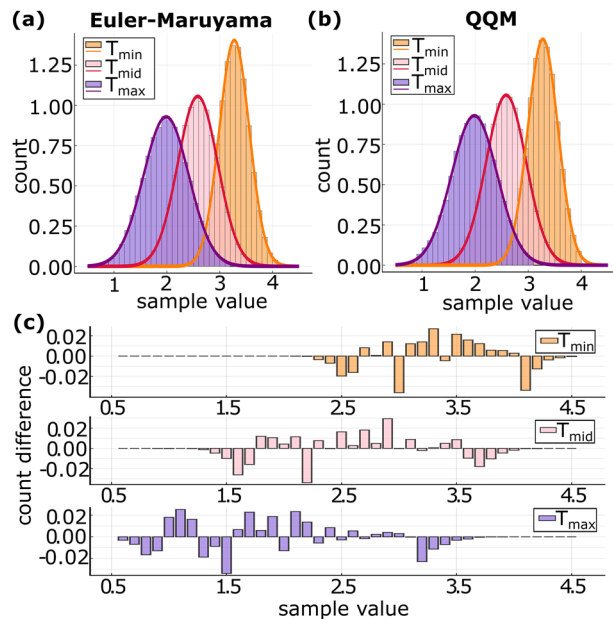


Figure 6. Comparison of histograms from the numerical SDE integration and QQM training. a) Distribution of samples from the Euler-Maruyama SDE solver (binned counts divided by the total number of samples N_s), shown against the analytic PDF at three time points T_{\min} ($t = 0$), T_{mid} ($t = 0.25$), and T_{\max} ($t = 0.5$). $N_s = 100,000$ samples are taken, and parameters are the same ($\nu = 1, \sigma = 0.7, x_0 = 4$). b) Distribution of samples generated from the DQC-based quantile function, for the training described in **Figure 5**. c) Bar height difference between (a,b) shown for T_{\min} (top), T_{mid} (middle), and T_{\max} (bottom).

Figure 6 for the same parameters as **Figure 5**. In **Figure 6a**, we show the three time slices of Euler-Maruyama trajectories, built with $N_s = 100000$ samples to see distributions in full. The counts are binned and normalized such that the total area of the histogram (bin width multiplied by bin height summed) is equal to one, and naturally show excellent correspondence with analytical results. The sampling from trained quantile function is performed by drawing random $z \sim \text{uniform}(-1, 1)$ for the same number of samples. In **Figure 6b**, we observe that QQM matches well the expected distributions. Importantly, the training correctly reproduces the widening of the distribution and the mean reversion, avoiding the mode collapse that hampers adversarial training.^[53,74] To further corroborate our findings, we plot the difference between two histograms (Euler-Maruyama and QQM) in **Figure 6c**, and observe that the count difference remains low at different time points.

3.1.3. Ornstein–Uhlenbeck with Data-Inferred Initial Condition

Next, we demonstrate the power of quantile function training from the available data (observations, measurements) corresponding to the Ornstein–Uhlenbeck process. Note that compared to the propagation of a known solution that is simplified by the boundary handling procedure, for this task we learn both the surface $G(z, t)$ and the initial quantile function $G(z, T_{\min})$. To learn the initial QF (same parameters as for **Figures 5** and **6**) we use QCL trained on observations. The observations are 100000 samples from the normal distribution with mean

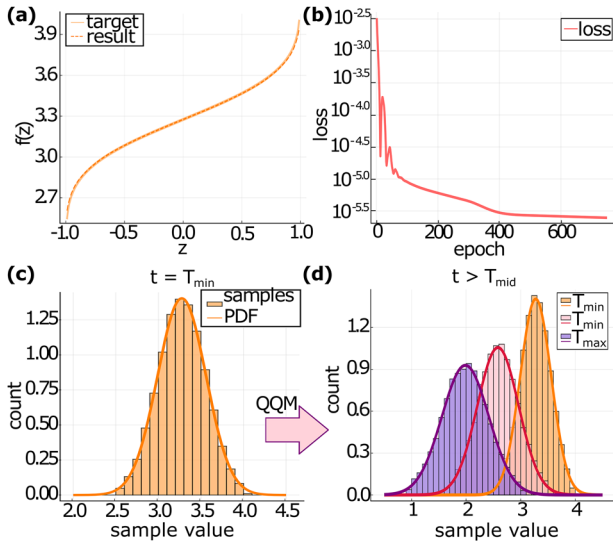


Figure 7. Quantile function trained on initial data. a) Trained QF for the Ornstein-Uhlenbeck process at $t = 0$ (dashed curve labeled as result), plotted together with the known true quantile (solid line labeled as target, results overlay). The parameters are the same as for Figure 5. b) Training loss at different epochs, with the final epoch producing the QF in (a). c) Normalized histogram of samples from the data-trained QF, plotted against the analytic distribution (PDF). $N_s = 100,000$ random samples are drawn and bin counts are normalized for a total area of one as before. d) Histograms for the data-trained QF evolved with quantum quantile mechanics, shown at three points of time.

$\mu_0 = x_0 \exp[\nu t_0]$ and standard deviation $\sigma_0 = \sqrt{\frac{\sigma^2}{\nu} (1 - \exp[2\nu t_0])}$, where $x_0 = 4$, $\nu = 1$, $t_0 = -0.2$, $\sigma = 0.7$ (all dimensionless units). The samples in the initial dataset are collected into bins and sorted in the ascending order as required by QF properties. From the original $N_s = 100000$ that are ordered we obtain an interpolated curve. We get target values for QCL training choosing $N_{\text{points}} = 43$ points in \mathcal{Z} between -1 and 1 . We note that the training set is significantly reduced, and such data-frugal training holds as long as the QF structure is captured (monotonic increase). The training points are in the Chebyshev grid arrangement as $\cos[(2n - 1)\pi / (2N_{\text{points}})]$ ($n = 1, 2, \dots, N_{\text{points}}$), this puts slight emphasis on training the distribution tails around $|z| \approx 1$. To make the feature map expressive enough that it captures full z -dependence for the trained initial QF, we use a tower-type product feature maps defined as $\widehat{U}_{\phi'}(z) = \bigotimes_{j=1}^N \exp[-i \arcsin(z)] \widehat{Z}_j / 2$, where rotation angles depend on the qubit number j . For the training we again use a six-qubit register, and follow the same variational strategy as in the previous subsection. We initialize the circuit using an initialization procedure as detailed in Appendix E. We observe that a high-quality solution with a loss of $\approx 10^{-6}$ for $G(z, 0)$ can be obtained at the number of epochs increased to few thousands, and we find that pre-training with product states allows reducing this number to hundreds with identical quality.

The results are shown in **Figure 7**, with the QF trained from data shown in Figure 7a by the dashed curve that overlays the target QF. The circuit converges to 10^{-6} loss level (Figure 7b), being close to the model expressivity limit of the feature map itself. Performing sample generation at the initial time, in Figure 7 we

observe good correspondence with the expected PDF (sampling procedure is the same as in Figure 6). At the same time we note that small deviations in trained QF (and its derivative) lead to significant deviations for statistics, stressing the importance of expressive circuits and stable training. Using the trained QF as the initial condition, we evolve the system as before (training details are presented in Appendix C). We perform generative modeling at later points of time T_{mid} and T_{max} . The histograms in Figure 7 confirm the high quality of sampling, and show that the approach is suitable for time series generation.

Let us now discuss the trainability of the proposed approach. As we deal with QNNs and non-convex optimization, we expect that at increasing system size (and the number of variational parameters) there can be regions with vanishing gradients, so-called barren plateaus. This is an issue that plagues many variational quantum algorithms. In our simulations, we do not experience signs of vanishing gradients, as we work with up to ten qubits with 190 parameters. It is likely that for our set up which uses a HEA this issue will be encountered as qubit number and ansatz depth increases. The question of avoiding barren plateaus is a current hot topic of research. One such direction is using alternative forms of ansatz such as convolutional neural networks^[98] or suppressing entanglement entropy of variational states.^[99] Alternatively, the overparametrization of the variational circuit is also a current topic of interest and a potential method to reduce the effects of barren plateaus.^[100]

Additionally, landscapes of variational quantum circuits may include multiple local minima.^[101] However, we do not search for a ground state, but sufficiently good approximation of the quantile function, and this can change the loss landscape. We also note that for random initialization we may get models where derivative terms overpower other terms, making the loss untrainable. However, this can be readily fixed by the initialization proposed in Appendix E, or going to another random set. From the point of expressivity, we note that quantile functions for continuous distributions are smooth. We expect that for sufficiently rich feature maps QFs can be represented to high precision.

3.2. qGAN-Based Generative Modeling

Generative adversarial networks represent one of the most successful strategies for generative modeling.^[53] It is used in various areas, ranging from the fashion industry to finance, where GANs are used to enrich financial datasets. The latter is specially relevant when working with relatively scarce or sensitive data. The structure of GAN is represented by two neural networks: a generator G_{NN} and a discriminator D_{NN} . The generator takes a random variable $z \sim p_z(z)$ from a latent probability distribution $p_z(z)$. This is typically chosen as a uniform (or normal) distribution for $z \in (-1, 1)$. Using a composition $g_L \circ \dots \circ g_2 \circ g_1$ of (non-linear) operations $\{g_i\}_{i=1}^L$ such that the generator prepares a fake sample $G_{\text{NN}}(z)$ from the generator's probability distribution p_G , $G_{\text{NN}}(z) \sim p_G(G_{\text{NN}}(z))$. The goal is to make samples $\{G_{\text{NN}}(z)\}_{s=1}^{N_s}$, as close to the training dataset as possible, in terms of their sample distributions. If true samples $x \sim p_{\text{data}}(x)$ are drawn from a (generally unknown) probability distribution $p_{\text{data}}(x)$, our goal is to match $p_G(G_{\text{NN}}(z)) \approx p_{\text{data}}(x)$. This is achieved by training the discriminator network D_{NN} to distinguish true from fake samples,

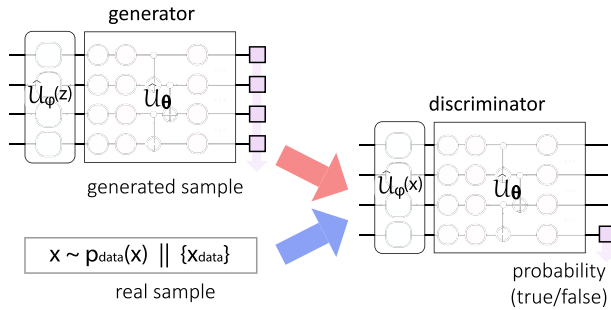


Figure 8. Quantum GAN workflow. The quantum circuits are used both for generative modeling at $t = T_{\min}$ (generator) and discrimination between real and fake samples (discriminator). The generator circuit $G_Q(z)$ is composed of the product feature map and HEA variational circuit. The discriminator $D_Q(x)$ is trained to distinguish samples from the initial data distribution.

while improving the quality of generated samples $\{G_{NN}(z)\}_{s=1}^{N_s}$, optimizing a minimax loss

$$\min_{G_{NN}} \max_{D_{NN}} \mathcal{L}_{GAN} = \min_{G_{NN}} \max_{D_{NN}} \left\{ \mathbb{E}_{x \sim p_{data}(x)} [\log D_{NN}(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D_{NN}(G_{NN}(z)))] \right\} \quad (11)$$

where D_{NN} and G_{NN} are the trainable functions represented by the discriminator and generator, respectively. The first loss term in Equation (11) represents the log-likelihood maximization that takes a *true* sample from the available dataset, and maximizes the probability for producing these samples by adjusting variational parameters. The second term trains G_{NN} to minimize the chance of being caught by the discriminator. We stress that the loss corresponds to a minimax game, therefore instead of training to find a minima we are instead training to find the Nash equilibrium.^[22] Most importantly, we note that $G_{NN}(z)$ is a function that converts a random sample $z \sim \text{uniform}(-1, 1)$ into a sample from the trained GAN distribution — therefore representing a *quantile-like function*. This is the connection we develop further using the qGAN training.

Quantum GANs follow the same ideology as their classical counterparts, but substitute the neural representation of the generator G_{NN} and/or discriminator D_{NN} by quantum neural networks. In the following, we denote them as G_Q and D_Q , respectively. The schedule of qGAN training and circuits are presented in **Figure 8**. To apply qGANs for the same task of OU process learning, we concentrate on a continuous qGAN that uses the feature map encoding.^[67,72] We follow the training strategy from Ref. [67]. We try to model the normal distribution with zero mean and standard deviation of 0.2. Both the discriminator and generator use $N = 6$ registers with the expressive Chebyshev tower feature map^[45] followed by $d = 6$ HEA ansatz of the form as in Figure 4. The readout for the generator uses the cost operator $\langle \hat{Z}_1 \rangle$, and the discriminator uses the same cost operator with post processing such that we readout $(\langle \hat{Z}_1 \rangle + 1)/2 \in [0, 1]$ modeling the probability. As before, we use Adam and train the qGAN for 2000 epochs using the loss function (11). Due to the minimax nature of the training, the loss oscillates and instead of reaching (global) optimum qGAN tries to reach the Nash equilibrium. Unlike QQM training, we cannot simply use variational param-

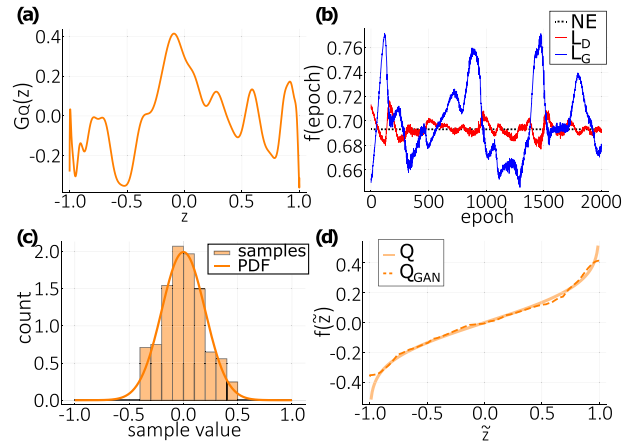


Figure 9. qGAN training and fixed-time sampling. a) Generator function is shown for the optimal training angles. b) Generator (L_G , red) and discriminator (L_D , blue) loss terms at different epochs. The Nash equilibrium at $-\ln(1/2)$ is shown by black dotted line (NE). c) Normalized histogram for qGAN sampling ($N_s = 100,000$), as compared to the target normal distribution ($\mu = 0, \sigma = 0.2$). d) Ordered quantile $Q_{GAN}(z)$ from the resulting qGAN generator shown in (a) (dashed curve), as compared to the true QF of the target distribution (solid curve).

eters for the final epoch, and instead test the quality throughout. To get the highest quality generator we test how close together are the discriminator (L_D) and generator (L_G) loss terms by measuring the mean square distance between the two values. If they are within $\epsilon = 0.1$ distance we perform the Kolmogorov–Smirnov (KS) test^[60] and check the distance between the currently generated samples and the training dataset. The result with minimal KS is chosen. We stress that KS is not used for training, and is exclusively for choosing the best result.

The results for qGAN training are shown in **Figure 9**. A total of 10 000 samples from $N(0, 0.2)$ are used as training data. A different random subset of 1000 samples is used at each epoch for the loss evaluation. The trained generator $G_Q(z)$ is shown as a function of the latent variable z . We note that it has a strongly-oscillating nature. The training is shown separately for the generator (blue curve) and discriminator (red curve) loss terms. They oscillate around the analytic value for the Nash equilibrium (black dotted line, NE), and briefly settle around NE after 1600 epochs where resultant circuit parameters are saved. We sample the qGAN generator using $N_s = 100000$ and plot the normalized histogram in Figure 9c. We observe that the distribution roughly matches the target (solid curve, PDF), though finer points are missing (cf. Figure 7c), including the missing tail at negative values. In Figure 9b, the loss is shown and exhibits the oscillations that are often seen in qGAN training due to the competition in the minimax loss, and associated mode collapse phenomena.^[102] There are further techniques that could be implemented to alter this such as Refs. [103, 104]. Naturally, the generator of qGAN $G_Q(z)$ does the same job as the trained quantile function $G(z)$ from previous subsections. We proceed to connect the two explicitly.

Reordered Quantile Functions and their DEs. The main difference between the quantile function and the generator of qGAN is that the true QF is a strictly monotonically increasing function, while the qGAN generator G_Q is not. We can connect them

by noticing that the qGAN works with the latent variable $z \in \mathcal{Z}$, which we can rearrange into a QF by ordering the observations and assigning them the *ordered* latent variable $\tilde{z} \in \tilde{\mathcal{Z}}$ (both functions produce the same sample distribution). It is convenient to define a mapping $h: \tilde{\mathcal{Z}} \rightarrow \mathcal{Z}$ for G_Q which rearranges it into increasing form, $Q_{GAN}(\tilde{z}) = G_Q(h(\tilde{z}))$. In practice, finding $h(\tilde{z})$ requires the evaluation of $G_Q(z) \forall z \in \mathcal{Z}$ and re-assigning the samples to values of $\tilde{\mathcal{Z}}$ in ascending order. Importantly, both h and its inverse $\text{inv}[h]: \mathcal{Z} \rightarrow \tilde{\mathcal{Z}}$ can be defined in this process. In Figure 9d, we show the results of reordering for the generator function in Figure 9a. The reordered quantile Q_{GAN} is plotted (dashed curve), approximately matching the target quantile (solid curve). We observe that the center of the quantile is relatively well approximated but the tails are not (particularly for $\tilde{z} < 0$). This agrees with what is observed in the sampling shown in Figure 9c. Having established the correspondence for qGAN-based generative modeling and quantile-based modeling we ask the question: can we apply differential equations to the quantile-like function to add differential constraints, and evolve the system in time enabling generative modeling?

4. Discussion

The answer to the question above is far from trivial. To use a reordered qGAN quantile function for further training and time-series generation we need to account for the mapping when writing differential equations of quantile mechanics. Let us look into a specific case to develop an intuition on the behavior of reordered quantile functions with differential equations. A quantile function $Q(\tilde{z})$ of a normal distribution with mean μ and standard deviation σ satisfies a quantile ODE^[81,82]

$$\frac{d^2 Q}{d\tilde{z}^2} - \frac{Q - \mu}{\sigma^2} \left(\frac{dQ}{d\tilde{z}} \right)^2 = 0 \quad (12)$$

where we use the tilde notation \tilde{z} to highlight that this is an ordered variable. Assuming perfect training such that $Q_{GAN}(\tilde{z}) = G_Q(h(\tilde{z}))$ closely matches $Q(\tilde{z})$, we substitute it into Equation (12), and observe that the original qGAN generator obeys

$$\begin{aligned} \frac{d^2 G_Q(z)}{dz^2} - \frac{G_Q(z) - \mu}{\sigma^2} \left(\frac{dG_Q(z)}{dz} \right)^2 \\ = \frac{\text{inv}[h]''(z) \frac{dG_Q(z)}{dz}}{\text{inv}[h]'(z)} \end{aligned} \quad (13)$$

The left-hand side (LHS) of Equation (13) has the same form as for the true QF [cf. Equation (12)], but the right-hand side (RHS) differs from zero and involves derivatives of the inverted mapping function $\text{inv}[h](z)$. This has important implications for training $G_Q(z)$ with differential constraints, as the loss term includes the difference between LHS and RHS. Let us analyze the example of the quantile ODE in Equation (13). In Figure 9a, we plot the LHS for $G_Q(z)$ coming from the qGAN training. The result is a smooth function, and we expect all relevant terms, including derivatives $dG_Q(z)/dz$ or $d^2 G_Q(z)/dz^2$, can be evaluated and trained at all points of the latent space. However, the problem arises when the RHS enters the picture. The additional term

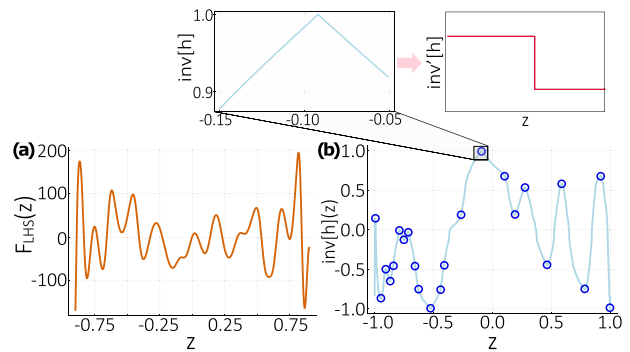


Figure 10. qGAN quantile function analysis. In (a), we plot the LHS of Equation (13) for the trained qGAN generator $G_Q(z)$ shown in Figure 9a. The resulting function is oscillatory but smooth. b) Inverse mapping function $\text{inv}[h]$ shown as a function of z . It transforms $G_Q(z)$ into the increasing quantile function $Q_{GAN}(\tilde{z})$ that is plotted in Figure 9d. We highlight the non-differentiable points by blue circles, and zoom in on the characteristic behavior in the inset above. The derivative is not defined at the discontinuity (top right inset).

strongly depends on the contributions coming from inverse map derivatives $\text{inv}[h]'$ and $\text{inv}[h]''$. At the same time we find that the map from a non-monotonic to a monotonically increasing function is based on a multivalued function (see discussion and examples in Appendix B). Furthermore the inverse of the map (along with the map itself) is continuous but not smooth—it becomes non-differentiable at some points due to $G_Q(z)$ oscillations. As an example in Figure 10b, we show $\text{inv}[h]$ from training in Figure 9, highlighting the points with non-analytic behavior blue circles. The inset for Figure 10b clearly shows the discontinuity. This translates to the absence of $\text{inv}[h]'(z)$ at a set of points, which unlike zero derivatives cannot be removed by reshuffling the terms in the loss function. The discovered unlikely property of the mapping puts in jeopardy the attempts to use differential-based learning for qGAN generators. While more studies are needed to estimate the severity of discontinuities (and if the set of such points can be excluded to yield stable training), our interim conclusion is that quantile functions in the canonical increasing form are more suitable for evolution and time series generation.

5. Conclusion

We proposed a distinct quantum algorithm for generative modeling from stochastic differential equations. Summarizing the findings, we have developed the understanding of generative modeling from stochastic differential equations based on the concept of quantile functions. We proposed to represent the quantile function with a trainable (neural) representation, which may be classical- or quantum-based. In particular, we focused on parameterizing the trainable quantile function with a differentiable quantum circuit that can learn from data and evolve in time as governed by quantile mechanics equations. Using Ornstein–Uhlenbeck as an example, we benchmark our approach and show that it gives a robust strategy for generative modeling in the NISQ setting. Furthermore, we notice that adversarial schemes as continuous qGAN lead to modified quantile-like function that potentially have intrinsic obstacles for evolving them in time.

For future implementation of this algorithm for more complex problems further work on choosing suitable feature maps that can accommodate the problem whilst remaining trainable is needed. One such approach could be encoding implicit biases^[105] and utilizing symmetries via geometric machine learning techniques.^[106,107] This is particularly suitable for quantile functions as they are monotonically increasing functions where derivatives can be constrained. We conclude by saying that the strategy we propose uses the large expressive power of quantum neural networks, and we expect elements of the approach can be used for other architectures.

Appendix A: Experimental Section

In this section, we describe the details of circuit differentiation and the proposed boundary handling procedure for multivariate functions.

Calculating Second-Order Derivatives

For solving SDEs, we need to access the derivatives of the circuits representing quantile functions dG/dz , d^2G/dz^2 , where z is a latent variable. This can be done using automatic differentiation techniques for quantum circuits, where for near-term devices and specific gates we can use the parameter shift rule.^[87,108,109] The differentiation of quantum feature maps follows the DQC strategy,^[45] where we estimate dG/dz as a sum of expectation values

$$\frac{dG(z)}{dz} = \frac{1}{2} \sum_{j=1}^N \varphi_j'(z) (\langle G_j^+ \rangle - \langle G_j^- \rangle) \quad (\text{A1})$$

where $\langle G_j^+ \rangle$ and $\langle G_j^- \rangle$ denote the evaluation of the circuit with the j -th gate parameter shifted positively and negatively by $\pi/2$. This generally requires $2N$ circuit evaluations. The (non-linear) function $\varphi_j(z)$ represents the z -dependent rotation phase for the j -th qubit. A popular choice is $\varphi_j(z) = \arcsin(z)$ (same for all qubits) referred as a product feature map, and other choices include tower feature maps.^[45]

Quantum circuit differentiation for higher-order derivatives was recently considered in several studies.^[109–111] Extending the feature map differentiation to the second order, we use the parameter shift rule alongside the product rule, and calculate d^2G/dz^2 as

$$\begin{aligned} \frac{d^2G(z)}{dz^2} &= \frac{1}{2} \sum_{j=1}^N \varphi_j''(z) (\langle G_j^+ \rangle - \langle G_j^- \rangle) \\ &+ \frac{1}{4} \sum_{j=1}^N \sum_{k=1}^N \varphi_j'(z) \varphi_k'(z) (\langle G_{jk}^{++} \rangle - \langle G_{jk}^{+-} \rangle - \langle G_{jk}^{-+} \rangle \\ &+ \langle G_{jk}^{--} \rangle) \end{aligned} \quad (\text{A2})$$

where $\langle G_{jk}^{++} \rangle$ ($\langle G_{jk}^{--} \rangle$) denotes the evaluation of the circuit with the rotation angles on qubits j and k are shifted positively (negatively) by $\pi/2$. Similarly, $\langle G_{jk}^{+-} \rangle$ ($\langle G_{jk}^{-+} \rangle$) are defined for shifts in opposite directions. If implemented naively, the derivative in Equation (A2) requires $2N + 4N^2$ evaluations of circuit expectation values. This can be reduced by making use of symmetries in the shifted expectation values and reusing/caching previously calculated values, that is, those for the function and its first-order derivative. The reduced number of *additional* circuit evaluations required for the calculation of the second derivative is $2N^2$.

Boundary Handling for PDEs

As we consider differential equations with more than one independent variable, we need to develop a strategy for implementing (handling) the boundary in this situation. Let us consider a function of n variables. We consider an initial condition of $f(t=0, z) = u_0(z)$, where z is a vector of $n-1$ independent variables, and the first variable usually corresponds to

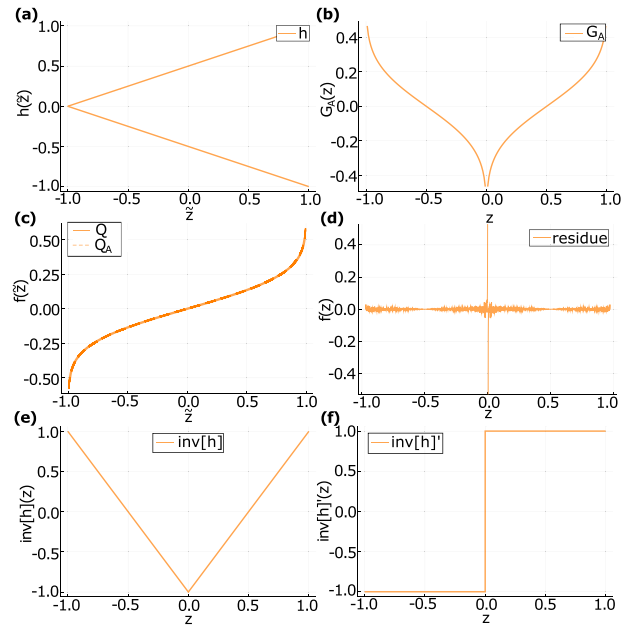


Figure B1. qGAN quantile reordering example. a) Mapping function $h(\tilde{z}) = \pm(\tilde{z} + 1)/2$. b) Non-ordered quantile function $G_A(z) = Q(\text{inv}[h](z))$ corresponding to mapping in (a) and normal quantile function. c) Known quantile function (solid curve, Q) compared to numerical ordering of G_A (dashed line, Q_A). d) RHS of Equation (A2) in the main text, plotted for G_A shown in (b). Derivatives of $\text{inv}[h]$ are calculated using finite differencing. e) Inverse mapping function $\text{inv}[h]$ plotted for different values of the latent variable z . We note the point of non-differentiability at $z = 0$. f) Analytic first derivative of $\text{inv}[h]$ that has a discontinuity at $z = 0$.

time. Here we extend several techniques, corresponding to pinned type and floating type boundary handling, previously considered for the single-variable case in Ref. [45].

When considering just one independent variable, a pinned boundary handling corresponds to encoding the function as $f(t) = G(t)$. The boundary is then ‘pinned’ into place by use of a boundary term in the loss function $\mathcal{L}^B = [f(t_0) - u_0]^2$. For multiple independent variables this generalizes to $f(t, z) = G(t, z)$ and $\mathcal{L}^B = \sum_i [f(t_0, z_i) - u(z_i)]^2$, where $\{z_i\}$ are the set of points along $t = 0$ at which the boundary is being pinned.

When using the floating boundary handling the boundary is implemented during the function encoding. In this case the function encoding is generalized from its single-variable representation, $f(t) = u_0 - G(0) + G(t)$, to the multivariate case as

$$f(t, z) = u_0(z) - G(0, z) + G(t, z) \quad (\text{A3})$$

This approach does not require the circuit-embedded boundary, but instead needs derivatives of $u_0(z)$ for calculating the derivatives of f with respect to any $z \in \mathcal{Z}$.

Appendix B: Analytic Quantile Reordering for qGAN Generators

To understand the reordering procedure for qGAN generators and strictly increasing quantile functions, let us consider a simple example.

We start by considering a quantile-like generator $G_A(z)$ and use the mapping h that reorders it into ideal quantile function (QF) for the normal distribution. The mapping reads $h(\tilde{z}) = \pm(\tilde{z} + 1)/2$, and is shown in Figure B1a as a multivalued function. It ensures that if we start from the normal QF, we arrive to $G_A(z)$ with a single dip. The corresponding qGAN-like generator $G_A(z)$ with a single dip is shown in Figure B1b (we consider

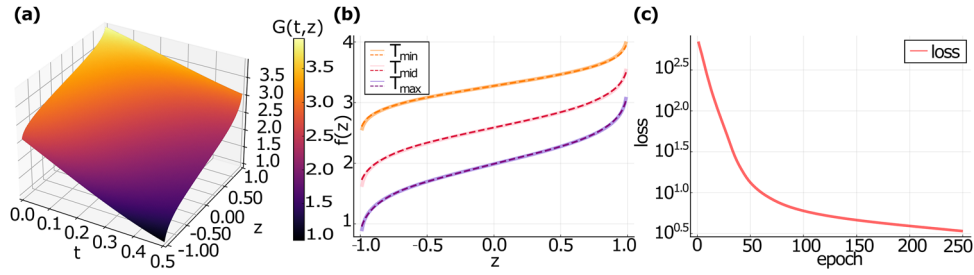


Figure C1. Time-evolution of Ornstein-Uhlenbeck process with data-inferred initial condition. The initial conditions is taken from Figure 7 and we use the same OU model parameters. a) Surface plot for the quantile function evolved in time. b) Quantile functions shown at three time points (being the same as in the main text). c) Loss as a function of epoch from training in (a,b).

$\mu = 0$ and $\sigma = 0.2$). Our motivation is to understand how the presence of nonmonotonicity changes the behavior of the system.

First, let us check that the reordering into increasing quantile function works as expected. Assigning the values of $G_A(z)$ in the ascending order we get the reordered QF $Q_A(\bar{z})$. This is plotted in Figure B1c, matching the ideal $Q(\bar{z})$ as expected. Once we have established the mapping for ideal re-ordering, let us look at its properties. In the results section, we discussed how the reordered quantile function from qGAN training matches the appropriate quantile ODE. We can perform the same check for the simple re-ordering presented above. Evaluating the difference between the RHS and LHS of Equation (13) (akin to loss term) for $G_A(z)$ and known mapping $h(\bar{z})$, we observe that the difference remains zero everywhere (we have a perfect solution), apart from the middle point $z = 0$ where it diverges (Figure B1d). For calculating the derivatives of $\text{inv}[h]$ we use finite difference (first-order forward Euler's method with a step of 10^{-5}), similarly for the qGAN case, thus observing small noise coming from numerical differentiation. The noise could be further reduced by considering alternative numerical differentiation schemes such as higher-order schemes or those more robust to step size such as the complex step rule.^[112] The reason behind the unfavorable loss term behavior can be tracked to the properties of the mapping function. We show the inverse mapping $\text{inv}[h]$ plotted in Figure B1e, and its derivative $\text{inv}[h]'$ is presented in Figure B1f. We see that $\text{inv}[h]$ has a point of non-differentiability at $z = 0$ and $\text{inv}[h]'$ is discontinuous there. This provides the intuition behind the divergence. When training the DE-based loss for Equation (13) with $z = 0$ included the loss becomes non-trainable. We stress that the same is observed for the non-ideal G_Q , where multiple non-differentiable points appear that we do not know in advance. The issue of developing efficient workflow for training $G_Q(z)$, also including the time dimension, remains an important area for the future research.

Appendix C: Time Evolution for Data-Inferred Quantile Function

In the Results Section of the main text, we detail how DQC is used to time evolve an analytic initial condition and how to learn the initial condition based on observations. We can also time evolve with DQC based on the observed initial quantile. The set up of the DQC is the same as when an analytic initial condition leading to Figure 5. The result of the training is shown in Figure C1, showing that the same quality of propagation is obtained.

Appendix D: Solving Reverse-Time Stochastic Differential Equations with QQM

Interesting connections between thermodynamics, machine learning, and image synthesis have been uncovered in recent years. Recently, Denoising Diffusion Probabilistic Models (DDPM)^[113] were shown to perform high quality image synthesis at state-of-the-art levels,^[114] sometimes better than other generative methods like GAN-approaches. Ref. [115] realized

that such discrete DDPMs can also be modeled as continuous processes using *reverse-time* SDEs when combined with ideas from score-based generative modeling.

Ref. [116] derived the reverse-time form of general stochastic differential equations, being

$$dX_t = \bar{f}(X_t, t)dt + g(X_t, t)dW_t \quad (D1)$$

where now a *modified* diffusion term \bar{f} is given by

$$\bar{f}(X_t, t) = f(X_t, t) - g^2(X_t, t)\nabla_X \log[p(x, t)] \quad (D2)$$

Ref. [115] proposed to solve such equations with general-purpose numerical methods such as Euler–Maruyama and stochastic Runge–Kutta methods, as well as using predictor-corrector samplers. We envisage solving the reverse-time Fokker–Planck equation corresponding to Equation (D1) instead, and more precisely its quantitized form, using the method described in this paper. We note that the reverse-time form (not to be confused with backward-Kolmogorov) actually looks the same, but is simply solved backward in time starting from a ‘final condition’ rather than an ‘initial condition’ data set.

As noted in Ref. [115], DDPM can be regarded as the discrete form of a stochastic differential equation (SDE). In Ref. [117], a general framework based on continuous energy-based generative models for time series forecasting is established. The training process at each step is composed of a time series feature extraction module and a conditional SDE based score matching module. The prediction can be achieved by solving reverse time SDE. The method is shown to achieve state-of-the-art results for multivariate time series forecasting on real-world datasets. These works imply that the method described here can be used for (multivariate) time-series forecasting and high-quality image synthesis.

Appendix E: Initialization of Variational Parameters

In this section, we describe the method of *parameter initialization* that we use to ensure a good starting function when implementing QCL or DQC. This is achieved by having a circuit structure where the variational circuit can be initialized to the identity operator and two layers of single-qubit rotation gates (which we refer to as the initialization layers). The parameters of the initialization layers can then be set to provide a good starting fit of the initial trial function to (an estimate of) the target function.

The circuit structure that we use is shown in Figure E1. The variational circuits are formed by a parameterized circuit unitary $\hat{U}_{a/b}(\theta_k)$ followed by the circuit with the adjoint structure but independently tuned set of variational angles, $\hat{U}_{a/b}^\dagger(\theta_{k'})$. We include variational circuits before and after the feature map to aid expressivity. For initialization we set $\theta_1 = \theta_2$ and $\theta_3 = \theta_4$. This leads to the variational circuits being initialized as identity. We stress that during training the parameters of these circuits are considered distinct and thus when updated by classical optimizer they do not remain equal.

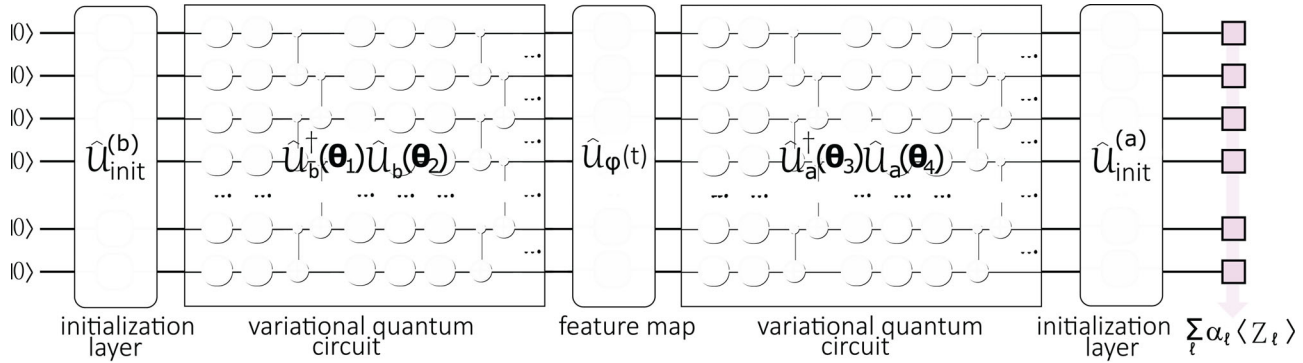


Figure E1. Initialization. We show the circuit structure used for when implementing initialization. It consists of layers of rotations known as the initialization layers $\hat{U}_{\text{init}}^{(b)}$, variational quantum circuits made up of $\hat{U}(\theta)$ and the feature map $\hat{U}_{\phi}(t)$. The variational quantum circuits are structured so that they can easily be initialized to the identity operator. Suitable parameters for $\hat{U}_{\text{init}}^{(b)}$ are chosen by performing classical function fitting. The feature map dictates which function form a basis for the fitting.

With the variational circuits initialized to identity the initial trial function is formed from measurements of the state received from the initialization layers and the feature map acting on the zero state. By considering a feature map which acts on each qubit individually as a product of Pauli rotations (as is the case with all feature maps discussed in this paper), and cost as the sum of individual qubit measurements, we need to treat only 1-local terms. This means that for initialization we have a circuit where the qubits are non-interacting and therefore the circuit is classically tractable. As the number of qubits N increase we can still describe the system with $\mathcal{O}(N)$ at the initialization stage, and we note that in principle k -local terms can also be included as long as the system remains tractable. Alternatively, we can also consider the circuit based on Clifford gates. In this case, the circuit remains classically tractable following the Gottesman–Knill theorem. For the remainder of this section we focus on the situation with initialization and feature map gates acting locally on single qubits.

With the circuit structure as discussed, the general form of the initial trial function can be expressed as

$$f_0(t) = \sum_{j=1}^N \alpha_j \left(c_j^{(1)}(\theta_{\text{init}}) g_j^{(1)}(t) + c_j^{(2)}(\theta_{\text{init}}) g_j^{(2)}(t) \right) \quad (\text{E1})$$

Here, $c_j(\theta_{\text{init}})$ are coefficients which depend on the angles parameterizing the initialization layer, α_j are the coefficients of the measurements in the cost function and $g_j(t)$ are functions of the variable encoded within the circuit by the feature map. Because this circuit is classically tractable the exact form of c_j and g_j remain calculable as N increases. The functions g_j come from the feature map encoding used. To see this note that for the feature maps considered the gate implemented on qubit j is

$$\hat{U}_j(t) = \hat{R}_{\beta_j}(\varphi_j(t)) = \exp\left(-i\hat{P}_j^{\beta} \varphi_j(t)/2\right) \quad (\text{E2})$$

$$= \cos(\varphi_j(t)/2)\hat{I}_j - i\sin(\varphi_j(t)/2)\hat{P}_j^{\beta} \quad (\text{E3})$$

where $\beta \in \{x, y, z\}$ are Pauli operators and φ_j are encoding functions. Therefore, the functions $\cos(\varphi_j(t)/2)$ and $\sin(\varphi_j(t)/2)$ are introduced into the state by the feature map. The feature map encoding is thus based on the sets of functions $g^{(1)} = \{\cos(\varphi_j(t))\}_j$ and $g^{(2)} = \{\sin(\varphi_j(t))\}_j$. Note that the coefficients in front depend on initialization layers, the feature map and the measurement operator chosen. For instance, by choosing specific circuit structures it is possible to select only a specific set of functions to use for initialization.

Knowing the form of the initial trial function we can choose θ_{init} and α_j such that it is a good starting state. We do this by performing classical regression. The fitting function's coefficient set $\{g_j\}$ are fitted to a limited

subset of the target function values. The size of $\{g_j\}$ is at most double the number of qubits and therefore when considering NISQ implementation will be relatively small. Therefore, as a low number of fitting functions are being fitted at a limited number of points, a low-order approximation and its associated coefficients can easily be found. These coefficients are then first used to choose α_j such that they are of suitable scale and the coefficient magnitudes desired are reachable by $\alpha_j c_j$. With α_j set, the expressions $\alpha_j c_j(\theta_{\text{init}})$ are then reversed to find the values which θ_{init} should be initialized to.

Once the initialization coefficients have been calculated the standard QCL procedure is then commenced. The initialization parameters will remain fixed whilst the variational ansatz parameters will be updated. As this happens the variational ansatz will no longer remain identity operators and more fitting functions will be introduced, with the set cardinality defined by the feature map. The circuit will no longer be classically tractable. The increase in the number of fitting functions (along with number of training points) leads to a better fit of the target data being possible. When using initialization a smaller learning rate is preferred to prevent immediate divergence from the initialized function.

We make use of initialization when training the quantile function on initial data as in Figure 7. We note however that using initialization is not a requirement for convergence. In Figure E2, the results of training the quantile function on initial data without initialization is shown. We can see that the loss value magnitude and the fit reached is similar to that achieved when training with initialization. The difference is seen in the number of epochs—without initialization more epochs are required to reach the same accuracy.

Appendix F: Scaling with Qubit Number

Qubit number is an important factor of model performance with number required depending on other factors such as the feature map, data reuploading and the problem considered. In general expressivity of the model scales with qubit number. Too few qubits can lead to low-expressivity which in turn leads to underfitting. Too many qubits can lead to overfitting and therefore generalize poorly. Furthermore more qubits leads to more quantum resource required. While there are generic guidances based on structural risk minimization,^[118] in practice the choice requires heuristics and understanding of possible implicit biases^[105] (an ongoing research direction in QML).

To see the performance of our proposed method as the qubit number scales we run the same calculation as used in Figure 5 but with varying number of qubits. As well as noting the final loss value from training we calculate the Kolmogorov–Smirnov (KS) test value for an out-of-training measure of model performance. The resulting loss and KS values are presented below.

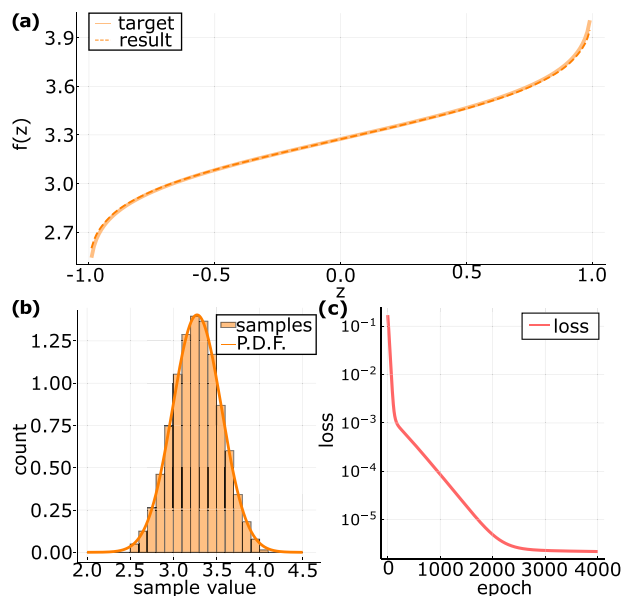


Figure E2. Quantile function trained on initial data — no initialization. a) Trained QF for the Ornstein–Uhlenbeck process at $t = 0$ (dashed curve labeled as result), plotted together with the known true quantile (solid line labeled as target, results overlay). b) Normalized histogram of samples from the data-trained QF, plotted against the analytic distribution (PDF). $N_s = 100000$ random samples are drawn and bin counts are normalized such that total area of the histogram is one. c) Training loss at different epochs, with the final epoch producing the QF in (a).

Qubit number	Loss	KS
2	9.40	1.117/0.276
4	0.88	0.371/0.046
6	1.08	0.356/0.025
8	0.94	0.320/0.022
10	1.52	0.376/0.046

We observe from the KS measure of performance that $N = 6$ and $N = 8$ -qubit models have the best performance for our model, while $N = 2, 4$ and 10 experience underfitting or overfitting. Furthermore the model remains trainable as qubit number increases. However, more iterations are required. For instance, 250 iterations are used for $N = 6$ and 500 for $N = 10$. Thus, for the given problem and chosen product feature map the middle ground is using the $N = 6$ embedding. In **Figure F1**, we show results as sampling histograms for the trained quantile function, at three time points when ten qubits are used.

Appendix G: Simulation of Noise in Training

Our results presented in the main text are simulated for an infinite number of shots. However, in physical implementations a limited number of shots is used leading to statistical noise in the read out. This typically follows a normal distribution. To check if our algorithm is robust in the presence of shot noise we perform an additional simulation, where noise is added to the function evaluation and its derivatives. Here we use the set up shown in **Figure 5**. We add noise to function evaluations following $\mathcal{N}(\mu = 0, \sigma = 0.05)$. We check that this is equivalent to the noise coming from function evaluations with 2000–3000 shots. No noise is added for measurements used for plotting as at this stage we are concerned with

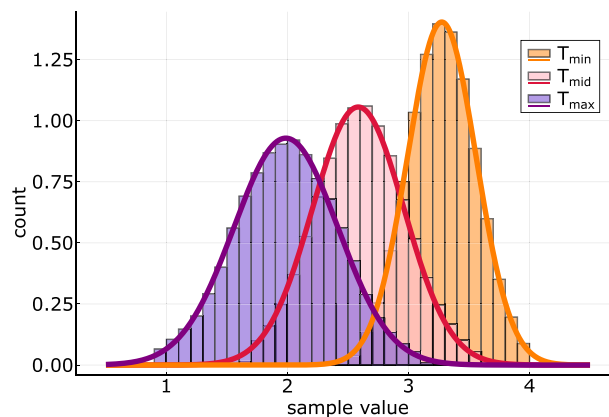


Figure F1. Time-evolution of Ornstein–Uhlenbeck process with ten qubits. Problem set up same as in **Figure 5** but with ten qubits and 500 iterations. Shown is the normalized histogram of samples from the data-trained QF, plotted against the analytic distribution (PDF). 100 000 random samples are drawn and bin counts are normalized such that the total area of each histogram is 1.

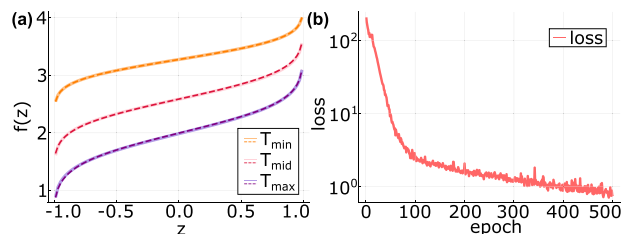


Figure G1. Time-evolution of Ornstein–Uhlenbeck process with simulated limited shots. Problem set up same as in **Figure 5** but with limited number of shots simulated by noise following $\mathcal{N}(\mu = 0, \sigma = 0.05)$ added to function evaluations during training. No noise added for final result plotting. a) Slice of resulting surface for quantile functions at three time points (same as in main text). b) Loss as a function of epoch from training resulting in (a).

the training. We check separately that the quantum circuit is successfully trained to represent the solution.

The results of this are shown in **Figure G1**. As can be seen a result with a similar level of accuracy is achieved, as compared to the case of no noise. Here more epochs are used (500 vs 250). Despite the loss being noisy (non-smooth) as function of epoch, we can recover a high quality solution.

Conflict of Interest

The authors declare no conflict of interest. A patent application for the method described in this manuscript has been submitted by Qu&Co with O.K., A.E.P., and V.E.E. as inventors.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Keywords

generative modeling, quantile function, quantum machine learning, quantum sampling

Received: March 16, 2023
Revised: June 22, 2023
Published online: August 7, 2023

- [1] C. W. Gardiner, P. Zoller, in *Quantum Noise*, Springer, Berlin/Heidelberg **2004**.
- [2] H.-P. Breuer, F. Petruccione, in *The Theory of Open Quantum Systems*, Oxford University Press, Oxford **2002**.
- [3] N. G. Van Kampen, *Phys. Rep.* **1976**, 24, 171.
- [4] B. Leimkuhler, C. Matthews, in *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*, Springer, Berlin **2015**.
- [5] D. D. Holm, *Proc. R. Soc. A* **2015**, 471, 20140963.
- [6] C. Cintolesi, E. Mémin, *Fluids* **2020**, 5, 108.
- [7] U. Dobramysl, M. Mobilia, M. Pleimling, U. C. Täuber, *J. Phys. A: Math. Theor.* **2018**, 51, 063001.
- [8] L. J. S. Allen, in *Mathematical Epidemiology*, Lecture Notes in Mathematics, (Eds.: F. Brauer, P. van den Driessche, J. Wu), Vol. 1945, Springer, Berlin, Heidelberg **2008**.
- [9] L. J. S. Allen, *Infect. Dis. Model.* **2017**, 2, 128.
- [10] Y. Rajabzadeh, A. H. Rezaie, H. Amindavar, *Digit. Signal Process.* **2016**, 54, 1.
- [11] B. Øksendal, in *Stochastic Differential Equations: An Introduction with Applications*, Springer, Berlin/Heidelberg **1998**.
- [12] T. Leung, L. Xin, in *Optimal Mean Reversion Trading: Mathematical Analysis and Practical Applications*, World Scientific, Singapore **2016**.
- [13] V. I. Bogachev, N. V. Krylov, M. Röckner, S. V. Shaposhnikov, in *Fokker–Planck–Kolmogorov Equations*, Vol. 207, Mathematical Surveys and Monographs, American Mathematical Society, Rhode Island, USA **2015**.
- [14] R. T. Q. Chen, D. Duvenaud, in *Advances in Neural Information Processing Systems*, **2019**, 32.
- [15] X. Li, T.-K. L. Wong, R. T. Q. Chen, D. K. Duvenaud, presented at *Proc. of The 2nd Sympos. on Adv. in Approximate Bayesian Inference*, *PMLR* **2020**, 118, 1.
- [16] P. E. Kloeden, E. Platen, in *Numerical Solution of Stochastic Differential Equations*, Springer, Berlin **1992**.
- [17] D. Zhang, L. Guo, G. E. Karniadakis, in *SIAM Journal on Scientific Computing* **2020**, 42, A639.
- [18] D. Zhang, L. Lu, L. Guo, G. E. Karniadakis, *J. Comp. Phys.* **2019**, 397, 108850.
- [19] M. A. Nabian, H. Meidani, *Prob. Eng. Mech.* **2019**, 57, 14.
- [20] L. Yang, D. Zhang, G. E. Karniadakis, in *SIAM Journal on Scientific Computing* **2020**.
- [21] P. Kidger, J. Foster, X. Li, H. Oberhauser, T. Lyons, in International conference on machine learning **2021**, 5453.
- [22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, **2020**, 63, 139.
- [23] S. L. Brunton, J. L. Proctor, J. N. Kutz, W. Bialek, *PNAS* **2016**, 113, 3932.
- [24] H. Schaeffer, *Proc. R. Soc. A* **2017**, 473, 20160446.
- [25] M. Raissi, in *The Journal of Machine Learning Research* **2018**, 19, 932.
- [26] M. Maslyayev, A. Hvatov, A. Kalyuzhnaya, *Proc. ICCS* **2019**, 11540, 635.
- [27] P. A. K. Reinbold, R. O. Grigoriev, *Phys. Rev. E* **2019**, 100, 1.
- [28] S. Aaronson, A. Arkhipov, *Theory of Computing*. **2013**, 9, 143.
- [29] A. P. Lund, M. J. Bremner, T. C. Ralph, *npj Quantum Inform.* **2017**, 3, 15.
- [30] J. M. Arrazola, P. Rebentrost, C. Weedbrook, in **2017**.
- [31] A. Deshpande, A. Mehta, T. Vincent, N. Quesada, M. Hinsche, M. Ioannou, L. Madsen, J. Lavoie, H. Qi, J. Eisert, D. Hangleiter, B. Fefferman, I. Dhand, in *Science advances* **2021**.
- [32] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, et al., *Nature* **2019**, 574, 505.
- [33] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, P. Hu, X.-Y. Yang, W.-J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N.-L. Liu, C.-Y. Lu, J.-W. Pan, *Science* **2020**, 370, 1460.
- [34] S. Chakrabarti, R. Krishnakumar, G. Mazzola, N. Stamatopoulos, S. Woerner, W. J. Zeng, *Quantum* **2021**, 5, 463.
- [35] D. J. Egger, C. Gambella, J. Marecek, S. McFaddin, M. Mevissen, R. Raymond, A. Simonetto, S. Woerner, E. Yndurain, *IEEE Trans. Quantum Eng.* **2020**, 1, 3101724.
- [36] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, *Nature* **2017**, 549, 195.
- [37] M. Benedetti, E. Lloyd, S. Sack, M. Fiorentini, *Quantum Sc. Technol.* **2019**, 4, 043001.
- [38] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, Xiao Yuan, L. Cincio, P. J. Coles, *Nat. Rev. Phys.* **2021**, 3, 625.
- [39] M. Schuld, N. Killoran, *Phys. Rev. Lett.* **2019**, 122, 040504.
- [40] A. Mari, T. R. Bromley, J. Izaac, M. Schuld, N. Killoran, *Quantum* **2020**, 4, 340.
- [41] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, S. Woerner, *Nat. Comput. Sci.* **2021**, 1, 403.
- [42] S. Y.-C. Chen, S. Yoo, *Entropy* **2021**, 23, 460.
- [43] S. Y.-C. Chen, T.-C. Wei, C. Zhang, H. Yu, S. Yoo, *Phys. rev. res.* **2022**, 4, 013231.
- [44] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, D. Jaksch, *Phys. Rev. A* **2020**, 101, 010301(R).
- [45] O. Kyriienko, A. E. Paine, V. E. Elfving, *Phys. Rev. A* **2021**, 103, 052416.
- [46] M. Knudsen, C. B. Mendl, in *Solving Differential Equations via Continuous-Variable Quantum Computers* arXiv:2012.12220, **2020**.
- [47] P. Garcia-Molina, J. Rodriguez-Mediavilla, J. J. Garcia-Ripoll, in arXiv:2104.02668 **2021**.
- [48] P. Rebentrost, B. Gupt, T. R. Bromley, *Phys. Rev. A* **2018**, 98, 022321.
- [49] N. Stamatopoulos, D. J. Egger, Yue Sun, C. Zoufal, R. Iten, N. Shen, S. Woerner, *Quantum* **2020**, 4, 291.
- [50] K. Kubo, Y. O. Nakagawa, S. Endo, S. Nagayama, *Phys. Rev. A* **2021**, 103, 052425.
- [51] J. Gonzalez-Conde, Á. Rodríguez-Rozas, E. Solano, M. Sanz, in arXiv:2101.04023 **2021**.
- [52] S. K. Radha, in arXiv:2101.04280 **2021**.
- [53] J. Gui, Z. Sun, Y. Wen, D. Tao, J. Ye, in *IEEE transactions on knowledge and data engineering* **2021**, 35, 3313.
- [54] S. Lloyd, C. Weedbrook, *Phys. Rev. Lett.* **2018**, 121, 040502.
- [55] P.-L. Dallaire-Demers, N. Killoran, *Phys. Rev. A* **2018**, 98, 012324.
- [56] J.-G. Liu, L. Wang, *Phys. Rev. A* **2018**, 98, 062324.
- [57] J. Zeng, Y. Wu, J.-G. Liu, L. Wang, J. Hu, *Phys. Rev. A* **2019**, 99, 052306.
- [58] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam, A. Perdomo-Ortiz, *npj Quantum Inform.* **2019**, 5, 45.
- [59] Y. Du, M.-H. Hsieh, D. Tao, in arXiv:1904.09602 **2019**.
- [60] C. Zoufal, A. Lucchi, S. Woerner, *npj Quantum Inform.* **2019**, 5, 45.
- [61] H. Situ, Z. He, Y. Wang, L. Li, S. Zheng, *Inform. Sci.* **2020**, 538, 193.
- [62] S. Y. Chang, S. Herbert, S. Vallecorsa, E. F. Combarro, R. Duncan, in *EPJ Web of Conferences* **2021**, 251.
- [63] M. Y. Niu, A. Zlokapa, M. Broughton, S. Boixo, M. Mohseni, V. Smelyanskiy, H. Neven, in *Entangling Quantum Generative Adversarial Networks* **2022**, 128, 2205.
- [64] M. Benedetti, E. Grant, L. Wossnig, S. Severini, *New J. Phys.* **2019**, 21, 043023.
- [65] P. Braccia, F. Caruso, L. Bianchi, *New J. Phys.* **2021**, 23, 053024.

- [66] H.-L. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu, R. Yang, T. Liu, M.-H. Hsieh, H. Deng, H. Rong, C.-Z. Peng, C.-Y. Lu, Y.-A. Chen, D. Tao, X. Zhu, J.-W. Pan, in *Physical Review Applied* **2021**, 16, 2, 024051.
- [67] J. Romero, A. Aspuru-Guzik, *Adv. Quantum Technol.* **2021**, 4, 2000003.
- [68] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, P. J. Coles, *Nat. Commun.* **2021**, 12, 1791.
- [69] B. Coyle, D. Mills, V. Danos, E. Kashefi, *npj Quantum Inf.* **2020**, 6, 60.
- [70] B. Coyle, M. Henderson, J. C. J. Le, N. Kumar, M. Paini, E. Kashefi, *Quantum Sci. Technol.* **2021**, 6, 024013.
- [71] A. Kondratyev, *Non-Differentiable Learning of Quantum Circuit Born Machine with Genetic Algorithm* Wilmott **2021**, 2021, 114, 5.
- [72] A. Anand, J. Romero, M. Degroote, A. Aspuru-Guzik, *Adv. Quantum Technol.* **2021**, 4, 2000069.
- [73] T. Goto, Q. H. Tran, K. Nakajima, in *Physical Review Letters* **2021**, 127, 9, 090506.
- [74] T. Li, S. Zhang, J. Xia, *Comput. Mater. Contin.* **2020**, 64, 401.
- [75] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytsky, R. Melko, *Phys. Rev. X* **2018**, 8, 021050.
- [76] Y. Du, M.-H. Hsieh, T. Liu, D. Tao, *Phys. Rev. Res.* **2020**, 2, 033125.
- [77] J. Liu, Ke-Thia Yao, F. Spedalieri, *Entropy* **2020**, 22, 1202.
- [78] Y. Takaki, K. Mitarai, M. Negoro, K. Fujii, M. Kitagawa, *Phys. Rev. A* **2021**, 103, 052414.
- [79] A patent application for the method described in this manuscript has been submitted by Qu&Co with OK, AEP and VEE as inventors.
- [80] J. P. Boyd, in *Solving Transcendental Equations: The Chebyshev Polynomial Proxy and Other Numerical Rootfinders, Perturbation Series, and Oracles*, Society for Industrial and Applied Mathematics, Philadelphia **2014**.
- [81] G. Steinbrecher, W. T. Shaw, *Eur. J. Appl. Math.* **2008**, 19, 87.
- [82] J. A. Carillo, G. Toscani, in *New Trends in Mathematical Physics*, World Scientific Publication, Hackensack, NJ **2004**, pp. 234–244.
- [83] W. T. Shaw, *J. Comput. Fin.* **2006**, 9, 37.
- [84] W. T. Shaw, T. Luu, N. Brickman, *Eur. J. Appl. Math.* **2014**, 25, 177.
- [85] W. Gilchrist, in *Statistical Modelling with Quantile Functions*, CRC Press, London **2000**.
- [86] K. Hornik, M. Tinchcombe, H. White, *Neur. Netw.* **1998**, 2, 359.
- [87] K. Mitarai, M. Negoro, M. Kitagawa, K. Fujii, *Phys. Rev. A* **2018**, 98, 032309.
- [88] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, J. M. Gambetta, *Nature* **2017**, 549, 242.
- [89] O. Kyriienko, A. E. Paine, V. E. Elfving, in *arXiv:2202.08253* **2022**.
- [90] N. Heim, A. Ghosh, O. Kyriienko, V. E. Elfving, in *arXiv:2111.06376*, **2021**.
- [91] J. Han, A. Jentzen, E. Weinan, *Proc. Natl. Acad. Sci.* **2018**, 34, 115.
- [92] O. Vasicek, *J. Finan. Econ.* **1977**, 5, 177.
- [93] R. S. Mamon, *Adv. Decis. Sci.* **2004**, 8, 131526.
- [94] J. Hull, A. White, *Rev. Financ. Stud.* **1990**, 3, 573.
- [95] M. Abramowitz, I. A. Stegun, in *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Government Printing Office, Washington, DC, USA **1972**.
- [96] X.-Z. Luo, J.-G. Liu, P. Zhang, L. Wang, *Quantum* **2020**, 4, 341.
- [97] C. Rackauckas, Q. Nie, *J. Open Res. Softw.* **2017**, 5, 15.
- [98] A. Pesah, M. Cerezo, S. Wang, T. Volkoff, A. T. Sornborger, P. J. Coles, *Phys. Rev. X* **2021**, 11, 041011.
- [99] S. H. Sack, R. A. Medina, A. A. Michailidis, R. Kueng, M. Serbyn, *PRX Quantum* **2022**, 3, 020365.
- [100] M. Larocca, N. Ju, D. García-Martín, P. J. Coles, M. Cerezo, in **2023**, 3, 542.
- [101] X. You, X. Wu, *PMLR* **2021**, 139, 12144.
- [102] J. Gui, Z. Sun, Y. Wen, D. Tao, J. Ye, *IEEE Trans. Knowl.* **2023**, 35, 3313.
- [103] B. T. Kiani, G. De Palma, M. Marvian, Z. Liu, S. Lloyd, *Quantum Sci. Technol.* **2022**, 7, 045002.
- [104] E. Bermot, C. Zoufal, M. Grossi, J. Schuhmacher, F. Tacchino, S. Vallecorsa, I. Tavernelli, in *Quantum Generative Adversarial Networks For Anomaly Detection In High Energy Physics* **2023**.
- [105] J. Bowles, V. J. Wright, M. Farkas, N. Killoran, M. Schuld, in *arXiv:2302.01365* **2023**.
- [106] M. Larocca, F. Sauvage, F. M. Sbahi, G. Verdon, P. J. Coles, M. Cerezo, *PRX Quantum* **2022**, 3, 030341.
- [107] J. J. Meyer, M. Mularski, E. Gil-Fuster, A. A. Mele, F. Arzani, A. Wilms, J. Eisert, *PRX Quantum* **2023**, 4, 010328.
- [108] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, N. Killoran, *Phys. Rev. A* **2019**, 99, 032331.
- [109] K. Mitarai, Y. O. Nakagawa, W. Mizukami, *Phys. Rev. Res.* **2020**, 2, 013129.
- [110] M. Cerezo, P. J. Coles, *Quantum Sci. Technol.* **2021**, 6, 035006.
- [111] A. Mari, T. R. Bromley, N. Killoran, *Phys. Rev. A* **2021**, 103, 012405.
- [112] J. R. R. A. Martins, P. Sturdza, J. J. Alonso, *ACM Trans. on Math. Soft.* **2003**, 29, 0098.
- [113] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, S. Ganguli, in *International conference on machine learning* **2015**.
- [114] J. Ho, A. Jain, P. Abbeel, in *Advances in neural information processing systems* **2020**, 33, 6840.
- [115] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, B. Poole, in *arXiv:2011.13456* **2020**.
- [116] B. D. O. Anderson, *Stoch. Proc. Appl.* **1982**, 12, 3.
- [117] T. Yan, H. Zhang, T. Zhou, Y. Zhan, Y. Xia, in *arXiv:2106.10121* **2021**.
- [118] M. C. Caro, E. Gil-Fuster, J. J. Meyer, J. Eisert, R. Sweke, *Quantum* **2021**, 5, 582.