

# Multi-variable integration with a variational quantum circuit

Juan M. Cruz-Martinez<sup>1</sup>, Matteo Robbiati<sup>1,2</sup>, and Stefano Carrazza<sup>1,2,3</sup>

<sup>1</sup>CERN, Theoretical Physics Department, CH-1211 Geneva 23, Switzerland.

<sup>2</sup>TIF Lab, Dipartimento di Fisica, Università degli Studi di Milano and INFN Sezione di Milano, Milan, Italy.

<sup>3</sup>Quantum Research Center, Technology Innovation Institute, Abu Dhabi, UAE.

**In this work we present a novel strategy to evaluate multi-variable integrals with quantum circuits. The procedure first encodes the integration variables into a parametric circuit. The obtained circuit is then derived with respect to the integration variables using the parameter shift rule technique. The observable representing the derivative is then used as the predictor of the target integrand function following a quantum machine learning approach. The integral is then estimated using the fundamental theorem of integral calculus by evaluating the original circuit. Embedding data according to a reuploading strategy, multi-dimensional variables can be easily encoded into the circuit's gates and then individually taken as targets while deriving the circuit. These techniques can be exploited to partially integrate a function or to quickly compute parametric integrands within the training hyperspace.**

## 1 Introduction

Many scientific and engineering problems require the evaluation of numerical integrals of varying complexity of the form:

$$I(\boldsymbol{\alpha}) = \int_{\mathbf{x}_a}^{\mathbf{x}_b} g(\boldsymbol{\alpha}; \mathbf{x}) d^n \mathbf{x}, \quad (1)$$

where the bold symbols correspond to vectors.

There are many numerical integration methods which tackle this problem, and the choice usually depends on the characteristics of the integrand functions. For instance, low-dimensional well-behaved integrals can be successfully integrated with quadrature methods. However, more complicated or higher-dimensional integrands will

lead to a significant increase in computational costs

In those cases, Monte Carlo (MC) methods are often favored due to their ability to handle a wider range of integrand functions without imposing stringent requirements and a convergence rate that does not depend on the dimensionality of the integrand. An important feature of MC methods is the possibility of binning partial results so that differential distributions in any of the integration variables can be obtained. However, in exchange for their flexibility, they suffer from slow convergence and require a large number of function evaluations. To mitigate these issues, various techniques have been proposed to speed up the integration process, reduce the number of integrand evaluations while producing accurate results [1–5].

In the context of particle physics, these advantages have made the VEGAS [6,7] MC algorithm into the gold standard for numerical integration. Over the past decade, there have been numerous attempts to further improve the algorithm without modifying the underlying strategy. Some examples are the implementation of the algorithm in new hardware devices [8,9], which offer a raw speed-up over traditional computing; the usage of multi-channel techniques [10], which exploit prior knowledge of the behavior of the different pieces of the integrand; or machine learning techniques to enhance the importance sampling algorithm [11,12].

However, all these methods suffer from the same drawback: obtaining a result requires the repeated numerical evaluation of the integrand in the region of interest. In addition, even if the integral smoothly depends on parameters which are not integrated over ( $\boldsymbol{\alpha}$  in Eq. (1)), any change in  $\boldsymbol{\alpha}$  requires a complete new run.

In Refs. [13,14] a new approach has been proposed which can be utilized to circumvent this

issue. These methods are based on using artificial Neural Networks (aNN) to build a surrogate model for the primitive of the integrand. Such a surrogate can keep the dependence on both the integration variables and the function parameters. By subsequently training the derivative of the model to approximate the integrand it is possible to recover its primitive. While the computational cost doesn't disappear (it is translated to the training of the process), it allows for the evaluation of the integral (or the marginalization of the integrand over any of the variables) for any choice of parameters within the training range at almost zero additional cost.

In this paper we apply the same strategy in a quantum machine learning (QML) [15–19] context to estimate the value of integrals of the form of Eq. (1).

The unique aspects of quantum circuits renders them an exceptional tool for this methodology. In the classical version, we need to train the derivative of the aNN. By taking the derivative, the architecture of the network can considerably change, possibly leading to a costly hyperparameter search in order to find the optimal model. With a quantum circuit instead, we can exploit the properties of quantum circuits in order to obtain the derivative using the Parameter Shift Rule (PSR) [20–23], therefore using the same architecture for the derivative and its primitive.

In doing this, we use `Qibo` [24–30], a full-stack and open-source framework for quantum simulation, control and calibration.

The exponential development of quantum technologies leads us to believe that quantum circuits and QML tools can be exploited to get faster and better performance with respect to the classical algorithms once the limited Near-Intermediate Scale Quantum [31] (NISQ) era is finally over. In particular, we note quite some interest on the field of High Energy Physics where many new algorithms are being developed and tested leading to a very robust ecosystem of quantum computing tools focusing on particle physics [32–38]

This paper is structured as follows, we expose the method in Sec. 2, after a brief introduction to QML and circuits derivative calculation respectively in Sec 2.1. and Sec. 2.3. In Sec. 3 we apply the method to two situations, a toy-model represented by a  $d$ -dimensional trigonometric function and a real-life scenario by calculating the inte-

gral of the  $u$  quark Parton Distribution Function (PDF).

All results can be reproduced using the code at:

<https://github.com/qiboteam/QiNNtegrate>.

## 2 Methodology

In this section we introduce well known concepts of quantum computation which are useful to better understand the methodology. Then, we describe the integration procedure more in detail.

### 2.1 Quantum Machine Learning in a nutshell

Classical ML is nowadays widely used to tackle statistical problems, such as classification, regression, density estimation, pattern recognition, etc. The goal of the ML algorithms is to teach a model to perform some specific task through an iterative optimization process. Let us recall here some basic ML concepts which equally apply to QML in order to establish the notation used through the paper.

We consider two variables: an input vector  $\mathbf{x}$  and an output vector  $\mathbf{y}$ , which is related to  $\mathbf{x}$  through some hidden law

$$\mathbf{y} = f(\mathbf{x}), \quad (2)$$

which we aim to estimate. These vectors can be composed of any number of variables and the dimensionality of the input and output data can in general be different. A parametric model  $\mathcal{M}(\boldsymbol{\theta})$  is then chosen to make predictions  $\mathbf{y}_{\text{est}}(\mathbf{x}|\boldsymbol{\theta}) = \mathcal{M}(\boldsymbol{\theta})\mathbf{x}$  of the output variables. Once a model is selected, a loss function  $J$  is typically used to verify the predictive goodness of the model  $\mathcal{M}$ . Finally, an optimizer is selected, which is in charge of computing:

$$\boldsymbol{\theta}_{\text{best}} = \operatorname{argmin}_{\boldsymbol{\theta}} \{J[\mathbf{y}_{\text{est}}(\mathbf{x}|\boldsymbol{\theta}), \mathbf{y}_{\text{meas}}]\}. \quad (3)$$

where  $\mathbf{y}_{\text{meas}}$  is a measured realization of  $\mathbf{y}$ .

In Quantum Computation (QC) we use two-level quantum systems called *qubits* to store information, and we act on their states with unitary operators we call *gates*, defining a totally reversible computation. These unitaries can be combined to build up more-than-one qubit gates, acting like quantum counterparts of the classical control gates. This paradigm enriches the computational possibilities of QC with new tools

such as superposition and entanglement, which can be used to define new computational models. In QML, Variational Quantum Circuits (VQC) [18, 39, 40] substitute the classical model  $\mathcal{M}(\boldsymbol{\theta})$  by a parametric circuit  $\mathcal{C}(\boldsymbol{\theta})$ . A VQC is a collection of gates which depend on a set of parameters  $\boldsymbol{\theta}$ . A circuit can be applied to an initial quantum state  $|\psi_i\rangle$  and, after the execution, a measurement can be performed on the final state  $|\psi_f\rangle$ , analogously to Eq. (2). We can collect information by executing the circuit  $N_{\text{shots}}$  times, and then calculating expected values of arbitrary chosen observables, such as:

$$G_{\hat{O}}(\boldsymbol{\theta}) = \langle \psi_i | \mathcal{C}^\dagger(\boldsymbol{\theta}) \hat{O} \mathcal{C}(\boldsymbol{\theta}) | \psi_i \rangle, \quad (4)$$

where  $\hat{O}$  is a target observable. Through this work  $\hat{O} = \hat{Z}$  with  $\hat{Z}$  a non-interacting  $\sigma_Z$ . In the following, for simplicity, we omit the reference to the observable.

We can use the QC tools as interpreters of the typical ML components to define a QML strategy. Several methods are known to embed input data  $\boldsymbol{x}$  into a quantum system [41–43]. In this work we follow the procedure presented in [42], which allows us to encode multi-dimensional variables into derivable gates of a circuit. Then, the model to be optimized will be interpreted by a parametric VQC. Depending on the presence or absence of a classical part in the model (*e.g.* a Neural Network), we can make distinction between hybrid classical-quantum or pure quantum QML architectures. Finally, the expected value of Eq. (4) can be identified with the predictions  $\boldsymbol{y}_{\text{est}}$ ,

$$\boldsymbol{y}_{\text{est}} = G(\boldsymbol{x}|\boldsymbol{\theta}) = \langle \psi_i | \mathcal{C}^\dagger(\boldsymbol{x}|\boldsymbol{\theta}) \hat{Z} \mathcal{C}(\boldsymbol{x}|\boldsymbol{\theta}) | \psi_i \rangle. \quad (5)$$

There are numerous possible choices for the loss function and the optimizer, depending on the type of model chosen and whether one is doing simulation or executing on a real quantum hardware. For example, by doing quantum simulation of a quantum neural network, a natural choice is to select a well known gradient-based optimizer [44–49] or some meta-heuristic algorithm like evolutionary strategies [50], simulated annealing [51], etc. Instead, when deploying the algorithm in actual quantum hardware it can be more effective to use shot-frugal optimizers [52–54] or to calculate gradients using metrics better suited to the QC context [55].

## 2.2 Circuit's ansatz

Building up our QML models, we encode the input data into the parametric gates of the circuit following the strategy suggested in [42], according to which an external variable can be uploaded into the angle  $\phi$  of rotational gates of the form:

$$R_k(\phi) = \exp\{-i\phi\hat{\sigma}_k\}, \quad (6)$$

where the hermitian generator of the rotation  $\hat{\sigma}_k$  is one of the Pauli's matrices.

In particular, we implement an architecture inspired by the uploading layer described in [42] called *fundamental Fourier Gate*,  $\mathcal{U}$ , which is composed of five sequential rotations around the  $z$  and the  $y$  axis. Our  $\mathcal{U}$  implementation is as follows:

$$\mathcal{U}(x|\boldsymbol{\theta}) = R_z(\theta_1)R_y(\theta_2)R_z(\theta_3)R_z(\theta_4 x)R_y(\theta_5), \quad (7)$$

where the data  $x$  is uploaded into the second gate in order of application on the initial state. The power of this approach lies in the fact that by re-uploading the data  $x$  into  $N$  consecutive channels in the form of Eq. (7), we approximate a target function as would an  $N$ -term Fourier series. This kind of strategy is introduced for a single qubit system on which a single variable is re-uploaded, but is easily extendible to a more-than-one qubit case. Moreover, increasing the number of qubits also increases the flexibility of the model, allowing us to upload different variables into different wires of the circuit. Several choices of architecture can be done, and we present here two of the various models implemented within the code accompanying this work.

The first one is shown in Fig. (1), we encode two dimensions in every qubit, such that the complexity of the circuit is equal to  $N_{\text{dim}}/2$ . Each uploading of the couple of variables  $(x_j, x_{j+1})$  into the associated qubit is in the form presented in Eq. (7).

Each family of gates  $\{\mathcal{U}_j, \mathcal{U}_{j+1}\}$  is then followed by an entangling channel  $\mathcal{W}_{\text{ent}}$ , which distributes the information accumulated by each qubit to the entire system. After the last layer, a final rotation  $R_y$  is added to each wire of the circuit before performing the measurements.

The second model implemented that we describe here aims to tackle the problem of fitting and integrating a Parton Distribution Function

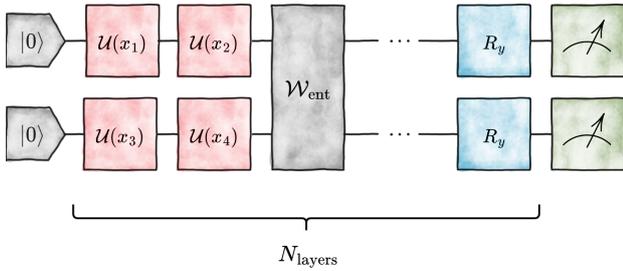


Figure 1: Diagram representation drawn with [56] of the reuploading ansatz used to fit a 4-dim function. The  $\mathcal{U}(x_i)$  quantum channel corresponds to the fundamental Fourier Gate presented in [42], while the entangling channel  $\mathcal{W}_{\text{ent}}$  is built with a combination of  $CZ$  gates.

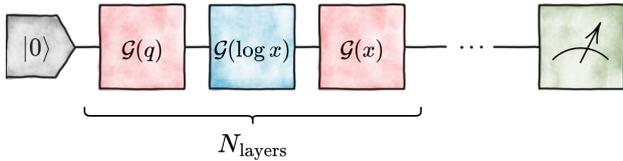


Figure 2: Schematic representation drawn with [56] of the qPDF ansatz used to fit the  $u$  quark parton density function.

(PDF) over  $x$  for any value of  $Q$ . More specifically, we define a model inspired by the one presented in [57] to fit the  $u$  quark PDF. We extend this model to fit the two dimensions  $x$  and  $q$  of the PDF. This second ansatz is shown in Fig. (2), where we define the following channels:

$$\begin{cases} \mathcal{G}(q) = R_y(\alpha_1 q + \beta_1), \\ \mathcal{G}(\log x) = R_z(\alpha_2 \log x + \beta_2), \\ \mathcal{G}(x) = R_y(\alpha_3 x + \beta_3), \end{cases} \quad (8)$$

with  $\alpha_i$  and  $\beta_i$  variational parameters. The imbalance of 2 : 1 in uploading  $x$  and  $q$  variables into the model is a choice motivated by the complexity of the PDF as the two target variables vary. In case of  $q$ , one upload gate is enough to tackle the problem. While the different behaviors of the PDF for small and large  $x$  requires uploading the variable twice (with the addition of a logarithmic activation function). This is one example of how a circuit designed to approximate a family of problems (in this case PDFs) can also be satisfactory exploited to integrate said family of functions thanks to the parameter shift rule, which is briefly described in the following section.

### 2.3 Derivative of a quantum circuit

Our aim is then to use the derivative of  $G$  defined in Eq. (4) with respect to  $\mathbf{x}$  as predictor of

the integrand function presented in Eq. (1). For this we use the Parameter Shift Rule (PSR) as the method for calculating the derivatives of  $G$  with respect to  $\mathbf{x}$ . The first example of PSR was presented in [17] and introduced a method for calculating the derivative of an expectation value in the form of Eq. (4) with respect to one of the rotation angles affecting the quantum circuit  $\mathcal{C}$ . We refer to a more general PSR formula presented in [20] and further developed later [21–23, 58].

According to [20], if a circuit depends on a parameter  $\mu \in \boldsymbol{\theta}$  through a single gate  $U$  whose hermitian generator has at most two eigenvalues, the derivative of  $G$  with respect to  $\mu$  can be exactly calculated as follows:

$$g(\mu) \equiv \partial_\mu G(\boldsymbol{\theta}) = r(G(\mu^+) - G(\mu^-)), \quad (9)$$

where  $\pm r$  are the eigenvalues of the generator of  $U$ ,  $\mu^\pm = \mu \pm s$  and  $s = \pi/4r$ . In this work we limit ourselves to rotational gates such as Eq. (6) for which  $r = \frac{1}{2}$  and  $s = \frac{\pi}{2}$ . The derivative of the circuit corresponds then to the execution of the same circuit twice per gate to which the input parameter has been uploaded to.

Since we never upload two input parameters to the same gate, the multidimensional extension is a trivial sequential application of the PSR per dimension and gate. If every dimension ( $d$ ) is uploaded once to every layer ( $l$ ), the total number of expectation values necessary is  $(2l)^d$ . Note that the number of input parameters does not need to coincide with the dimensionality of the integral, making this method particularly useful for parametric integrals which are less prone to the so-called curse of dimensionality.

### 2.4 Solving integrals with quantum circuits

In the following section we describe the QML training procedure and, once the optimization is done, how the final model can be used to calculate the integral of the target function.

Calculating the derivative of the circuit with the PSR, we are able to use the same architecture to evaluate the primitive of a function  $G$  and any of its derivatives  $g$ . To be more explicit, if we recall the formula with which we started the paper:

$$I(\boldsymbol{\alpha}) = \int_{\mathbf{x}_a}^{\mathbf{x}_b} g(\boldsymbol{\alpha}; \mathbf{x}) d^n \mathbf{x}, \quad (10)$$

the finite integral  $I(\boldsymbol{\alpha})$  can also be calculated through the fundamental theorem of the integral

calculus which for a 1-dimensional integral reads:

$$I(\boldsymbol{\alpha}) = G(x_b; \boldsymbol{\alpha}) - G(x_a; \boldsymbol{\alpha}), \quad (11)$$

with

$$G(x; \boldsymbol{\alpha}) = \int g(\boldsymbol{\alpha}; x) dx. \quad (12)$$

Our goal will be training the derivative of a VQC such that it approximates the function  $g$  at any point  $\mathbf{x}_j$  within the integration limits  $(\mathbf{x}_a, \mathbf{x}_b)$ .

$$g_{j,\text{est}}(\boldsymbol{\alpha}; \mathbf{x}_j | \boldsymbol{\theta}) = \left. \frac{\partial G(\boldsymbol{\alpha}, x_1, \dots, x_n | \boldsymbol{\theta})}{\partial x_1 \dots \partial x_n} \right|_{\mathbf{x}_j}. \quad (13)$$

If we have a way of evaluating  $g(\boldsymbol{\alpha}; \mathbf{x})$  or have access to measurements of its value we can generate a set of  $N_{\text{train}}$  training data. Once the predictions  $\{g_{j,\text{est}}\}_{j=1}^{N_{\text{train}}}$  are calculated for all the training data, we quantify the goodness of our model by evaluating a Mean-Squared Error loss function:

$$J_{\text{mse}} = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left[ g_{j,\text{meas}} - g_{j,\text{est}}(\boldsymbol{\alpha}; \mathbf{x}_j | \boldsymbol{\theta}) \right]^2, \quad (14)$$

where we indicate with the index ( $j$ ) the  $j$ -th element in the training dataset and its associated integrand value. The training is thus performed by iteratively updating the parameters  $\boldsymbol{\theta}$  in order to minimize Eq. (14). Various optimizers have been tested, including Powell method [59], L-BFGS [60], a Covariance Matrix Adaptation Evolutionary Strategy [50] (CMA-ES) and a Basin-Hopping algorithm [61]. The best results are obtained using L-BFGS and the Basin-Hopping optimizers when exact simulation is performed. On the contrary, we noticed that when shot-noise simulation is executed, the meta-heuristic algorithms (CMA-ES, Basin-Hopping) obtain better results.

Once the circuit's parameters are optimized, we have a fixed architecture which can be used to evaluate integrals with respect to any combination of the target variables. This aspect makes the strategy particularly interesting when dealing with high-dimensional functions. As an example, we can marginalize the integrand previously defined over the variable  $x_k$

$$I_{ab}(\dots, x_{k-1}, x_{k+1}, \dots) = \int_{x_{k,a}}^{x_{k,b}} g(\mathbf{x}) dx_k, \quad (15)$$

by uploading the integration limits,  $x_{k,a}$  and  $x_{k,b}$  and removing only that derivative from Eq. (13):

$$I_{ab}(\dots, x_{k-1}, x_{k+1}, \dots) \simeq g_{\text{est}}(x_{k,b} | \boldsymbol{\theta}_{\text{best}}) - g_{\text{est}}(x_{k,a} | \boldsymbol{\theta}_{\text{best}}), \quad (16)$$

where we write  $g_{\text{est}}$  explicitly depending only on  $x_k$  for simplicity. This can easily be extended for the partial integration of any of the variables of which  $I$  depends on.

### 3 Results

In order to showcase the possibilities of the methodology presented in this paper we are going to use a VQC for two different target functions. We will first show the flexibility of the method to obtain total or partial integrals and differential distributions, and then we will apply to a practical case in which the approach can introduce a net-gain. Both examples are implemented in the public code which accompanies this paper (among other examples).

#### 3.1 Toy Model

Our first example integrand is a d-dimensional trigonometric function:

$$g(\mathbf{x}) = \cos(\boldsymbol{\alpha} \cdot \mathbf{x} + \alpha_0), \quad (17)$$

with  $\mathbf{x}$  and  $\boldsymbol{\alpha}$  n-dim vectors. The integral of Eq. (17), while trivial to perform analytically, will serve to demonstrate how training one single circuit to obtain the primitive,

$$I(\boldsymbol{\alpha}; \mathbf{x}) = \int g(\boldsymbol{\alpha}; \mathbf{x}) d\mathbf{x}, \quad (18)$$

can provide us the flexibility to obtain other derived quantities.

For instance, we might be interested on the differential distributions  $\frac{dI(\boldsymbol{\alpha}; \mathbf{x})}{dx_i}$  for a given  $i$  and for different values of one of the parameters  $\boldsymbol{\alpha}$ . In general, this would require to perform the numerical integration once per choice of  $i$ , per bin in the distribution and choice of  $\boldsymbol{\alpha}$ . By having a surrogate for  $I(\boldsymbol{\alpha}; \mathbf{x})$  we can analytically obtain each distribution as seen in Fig. 3, were we collect results obtained by training the model with exact state vector simulation of the quantum circuits.

In Fig. 3 we have plotted the differential distribution

$$\frac{dI(\boldsymbol{\alpha}; x_1)}{dx_1}, \quad (19)$$

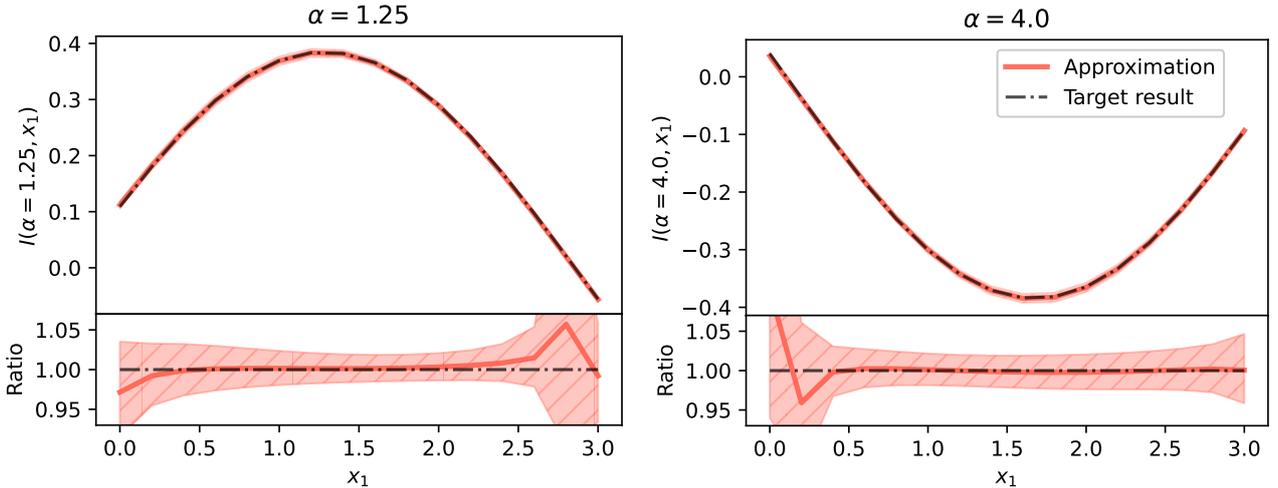


Figure 3: Differential distribution of the integral of Eq. (17) on  $x_1$  for different values of  $\alpha_0$  with  $\alpha = \{1, 2, \frac{1}{2}\}$ . The ranges chosen for  $x_1$  are also the ranges used for the integration of all other variables. These results are obtained through exact state vector simulation. The training was performed with 100 points in  $x$  and 10 different values of  $\alpha_0 \in (0, 5)$ , using the L-BFGS optimizer for 200-300 iterations. The error bands are computed retraining the circuit for different seeds.

for two different values of  $\alpha_0$  within the training range  $(0, 5)$ . All other parameters have remain fixed in order to minimize the computing cost in this toy-model example. The training range for the integrated variables  $(x_1, x_2, x_3)$  has been  $(0, 3.5)$ . In the plots we choose to integrate  $x_2$  and  $x_3$  from 0 to 3 for every value fo  $x_1$ , but any choice of integration limits within the integration range would be possible.

A shortcoming of this approach is the lack of an uncertainty associated to the numerical integration. In Ref. [14] the suggestion is to use an ensemble of replicas of the network trained to the same data in order to use the variance as an error. We have followed the same strategy here by training an ensemble of circuits randomizing the choice of training points which leads to a spread of the results.

Other numerical methods provide some shortcuts to obtain similar results. For instance MC integration methods would allow us to bin quantities which depend on the integration variables (provided that we know beforehand the distributions that we want to obtain). However, a change in the parameter  $\alpha_0$  will always lead to a new integration. Instead, once we have a circuit that approximates Eq. (18), any derived quantity in the training range is accessible without any new runs.

It is important to remark that there is no free lunch, we are paying the penalty in terms of eval-

uations of the integrand (and the surrogate) during the training, but the outcome is a flexible representation of the final quantity which can then be reutilized. In a similar manner to Monte Carlo methods, the accuracy of the calculation can be improved by increasing computational cost with either a larger number of samples or longer training lengths. Note that, unlike other machine learning problems, in this case we have a function that enables us to generate an unlimited amount of data for arbitrary inputs.

### 3.2 The $u$ -quark PDF

In the previous section we have chosen an ideal scenario with a function for which we know the primitive and against which we can exactly test. In what follows we consider an actual use-case for the approach that we propose in this paper: the integration of a function which is only known numerically and for which we have a representation in the form of a VQC.

In Ref. [57] a VQC was used to perform a PDF fit. This corresponds to a function  $f(x, Q)$  for  $Q$  fixed (called fitting scale). Obtaining the integral over  $x$  of the PDF is a necessary step of any proper determination in order to obtain normalized results and, due to the empirical nature of the function, can only be done numerically.

We use the ansatz described in Sec. 2.2 to obtain a model by which we can produce both the

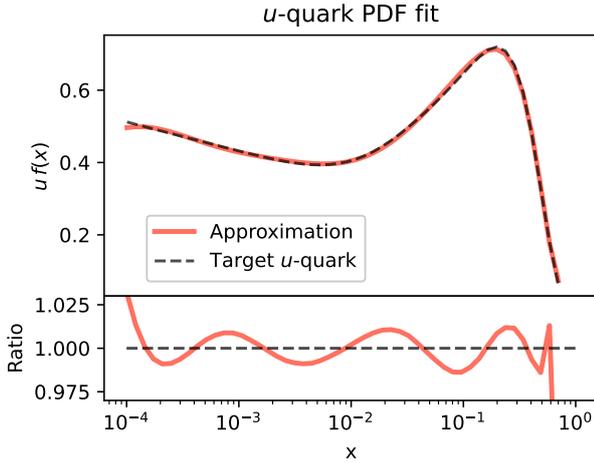


Figure 4: Comparison between the interpolation grid for the  $u$ -quark of the central replica of NNPDF4.0 and our fit, for a fixed value of  $Q = 1.67$ . While the ansatz in Sec. 2.2 gives us a model for the integral, the results training is performed onto the derivative. The training set uses  $\mathcal{O}(100)$  points for each fixed value of  $Q$ . These results are obtained through exact state vector simulation.

$u$ -quark PDF and its own integral, hence allowing for a prediction normalized by construction.

We train the model with the L-BFGS optimizer and performing exact simulation using `Qibo`. As the training data we utilize directly the  $u$ -quark from the NNPDF4.0 [62] PDF grids. We train in both  $x$  (the variable we need to integrate over) and  $Q$  to demonstrate the technique would work independently of the fitting scale.

We limit the training to  $(1e-4, 0.7)$  which is comparable with the ranges of data available in PDF determinations. In Fig. 4 we show the result of fitting the derivative of the ansatz to the training data, obtaining a very good description across the entire range.

Since the derivative of the ansatz is able to approximate the circuit, the associated integral,

$$I_u(Q^2) = \int_{10^{-4}}^{0.7} x u(x, Q) dx, \quad (20)$$

corresponds to the normalization for the  $u$ -quark. In order to show the generalizability of the method we have also trained the circuit for varying values of  $Q$ . The range of  $Q$  has been chosen to avoid crossing quark thresholds for which the matching between different  $n_f$  regions in NNPDF4.0 introduces numerical instabilities.

In QML, in real-life scenarios, one needs to account for the probabilistic shot-noise associated

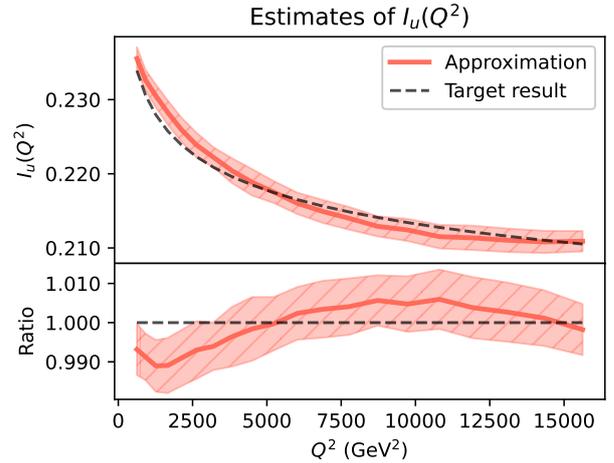


Figure 5: Above, integral of  $x u(x, q)$  calculated for  $N_q = 20$  values of  $q$ . These results are obtained by simulating circuits with shot-noise. In particular, for each  $q$  we perform  $N_{\text{runs}} = 100$  predictions and each prediction is obtained by executing the circuit  $N_{\text{shots}} = 10^6$  times. The orange line and its confidence belt are calculated using mean and one standard deviation over the  $N_{\text{runs}}$  prediction sets. Below, average relative percentage error calculated using the  $N_{\text{runs}}$  predictions. The training has been performed on  $\mathcal{O}(100)$  different values of  $Q$  evenly spaced on  $Q^2$  and took  $\mathcal{O}(20h)$  in a 32-cores machine.

with the measurement of the quantum states and the noise associated to the hardware. While the uncertainty associated to the training shown in Fig. 3 can in principle be reduced by increasing the training length, these uncertainties are intrinsic to the methodology.

In Fig. 5 we show the calculation of the integral of Eq. (20) for different values of  $Q$  within the training range. The circuits are simulated with shot-noise since we perform  $N_{\text{shots}} = 10^6$  to compute each expected value (circuit is called  $N_{\text{shots}}$  times for each estimation of the primitive  $G$ ). We then repeat every measurement of the integration  $N_{\text{runs}} = 100$  times. The error is computed by taking the variance of the measurement. The shaded band in Fig. 5 correspond to the  $1\sigma$  band.

This corresponds to an ideal real-life scenario since we are not considering hardware noise. The shot-noise scales as  $\frac{1}{\sqrt{N_{\text{runs}}}}$  and leads in this case to an uncertainty of about 1%. Due to the computational cost we don't include in this case the training uncertainty.

### 3.3 Normalized by construction

A possible application of our method to an actual physics problem is the determination of PDFs using quantum computers [57]. We leave the actual implementation to future work, but we outline here the methodology and expected gains.

In Refs. [62, 63] the normalized valence PDFs are implemented as:

$$V(x) = \frac{3\hat{V}(x)}{\int_{x_a}^{x_b} dx \hat{V}(x)}, \quad (21)$$

where the integral in the denominator is computed numerically between the extremes  $x_a$  and  $x_b$  and the unnormalized distribution  $\hat{V}$  is approximated by an aNN. Consequently, obtaining one single value for  $V(x)$  requires a costly numerical integral ( $\mathcal{O}(10^3)$  function calls).

With the `QiNNtegrate` approach we can instead construct a PDF which is already normalized:

$$V(x) = \frac{\hat{V}}{I_{\hat{V}}} = \frac{3 \sum_s^{\text{shifts}} G(x_s)}{G(x_b) - G(x_a)}, \quad (22)$$

where we have exchanged the computational burden of computing the numerical integrand by a costlier evaluation of the unnormalized distribution. The shifts in the sum corresponds to all pairs of  $(x_+, x_-)$  needed to compute the derivative as per Eq. (9).

For the ansatz used in this work we would need  $\mathcal{O}(10)$  circuit executions to estimate the derivative but have removed  $\mathcal{O}(10^3)$  executions from the training process. Furthermore, the fit would in this case benefit from a simpler functional form.

In addition, while in our naive approach each appearance of the VQC ( $G$ ) corresponds to a separate evaluation, novel proposals for non-demolition measurements involving gradients [64] could further enhance the methodology.

### 3.4 Integrating on a real qubit

In this section we present some results obtained executing this algorithm on a real quantum hardware. In particular, we use a superconducting device composed of a single qubit hosted at the Quantum Research Center (QRC) of the Technology Innovation Institute (TII). The entire process is realized using the `Qibo` [24–30] ecosystem;

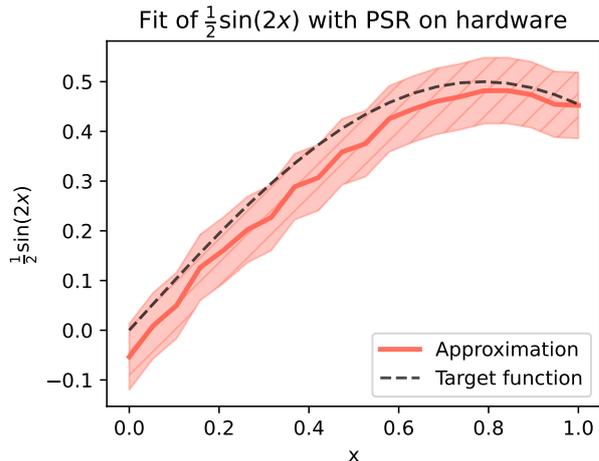


Figure 6: Estimates of the integrand  $g(x) = \frac{1}{2} \sin(2x)$  between  $x = 0$  and  $x = 1$  obtained by executing the presented algorithm on a real superconducting qubit. The target  $N_{\text{data}} = 20$  function values (black dashed line) are compared with the estimates (orange line), obtained as the average of  $N_{\text{runs}} = 5$  sets of predictions. The confidence interval is drawn using  $2\sigma$  error over the experiments. The results are computed without any kind of error mitigation technique.

the high level code is written with `Qibo` and then executed on the qubit by `Qibolab` [29]. In case we make use of `Qibosoq` [65], which is the server that integrates `Qick` [66] in the `Qibolab` ecosystem for executing arbitrary circuits and pulse sequences through RFsoC FPGA boards. All the single qubit characterization and calibration routines are performed using `Qibocal` [27, 30].

We tackle a simple example to reduce the number of expected values of the form of Eq. (4) to be evaluated. In fact, since the derivative of a circuit is used as predictor in our model,  $\mathcal{O}(2 * N_x)$  expected values are needed to compute each estimation, where  $N_x$  is the number of times  $x$  is uploaded into the model. For example, in case of the one dimensional  $u$  quark PDF presented in Sec. 3.2  $N_x = 2 N_{\text{layers}}$ . We present in Fig. 6 predictions of  $N_{\text{data}} = 20$  values of the integrand  $g(x) = \frac{1}{2} \sin(2x)$ , considering  $N_{\text{data}}$  values of  $x$  uniformly distributed in  $[0, 1]$ . We calculate  $N_{\text{runs}} = 5$  times the prediction for each  $x$  and use the mean and  $2\sigma$  to define the confidence intervals around the estimates which are shown in Fig. 6. During the process, each expectation value is obtained executing the circuit  $N_{\text{shots}} = 5000$  times. This simple example proves a simple integrand function can be fitted on a NISQ device.

As second test, we calculate the integral value of the target function:

$$I_{\text{target}} = \int_0^1 \frac{1}{2} \sin(2x) dx \quad (23)$$

$N_{\text{int}} = 10$  times obtaining as estimate:  $\hat{I}_{\text{target}} = 0.326 \pm 0.011$ , to be compared with the exact value  $I_{\text{target}} = 0.354$ . The error on the estimate is the standard deviation over the  $N_{\text{int}}$  results. We believe having such a satisfactory result even with noisy hardware can be motivated by the nature of the problem we are tackling. In fact, the target integral is here calculated following Eq. (11), and the difference between estimations can help in removing systematic errors that may occur when dealing with NISQ devices.

## 4 Conclusion

In this paper we have extended the methods proposed in Refs. [13, 14]. to quantum computers and shown how the properties of these new type of devices can introduce a practical advantage compared to classical alternatives by exploiting the properties of the PSR.

Furthermore, we have demonstrated a practical-case in which one can obtain a net advantage by utilizing this approach. A natural extension of this work is an enhancement of the methodology proposed in Ref. [57] in which the PDF fit could not be normalized due to technical constraints.

We then reported interesting results obtained on a real superconducting qubit, showing how a NISQ device can already be used to fit integrand functions following our algorithm. We have also made all code available in a public python framework `QiNNtegrate`<sup>1</sup> which can be used to reproduce the results of this work and which can be extended to other custom functions. `QiNNtegrate` is based on `Qibo` and thus the generated circuits can be either simulated in a classical computer or directly executed on hardware.

## Acknowledgments

We thank D. Maitre for the careful reading of the manuscript and many very useful comments. This project is supported by CERN's Quantum

Technology Initiative (QTI). MR is supported by CERN doctoral program. SC thanks the TH hospitality during the elaboration of this manuscript.

## References

- [1] Nicholas Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.
- [2] Russel E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 7:1–49, 1998.
- [3] Huicong Zhong and Xiaobing Feng. An efficient and fast sparse grid algorithm for high-dimensional numerical integration, 2022.
- [4] Zoubin Ghahramani and Carl Rasmussen. Bayesian monte carlo. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.
- [5] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
- [6] G. Peter Lepage. A New Algorithm for Adaptive Multidimensional Integration. *J. Comput. Phys.*, 27:192, 1978.
- [7] G. Peter Lepage. Adaptive multidimensional integration: VEGAS enhanced. *J. Comput. Phys.*, 439:110386, 2021.
- [8] Stefano Carrazza and Juan M. Cruz-Martinez. VegasFlow: accelerating Monte Carlo simulation across multiple hardware platforms. *Comput. Phys. Commun.*, 254:107376, 2020.
- [9] Pablo Gómez, Håvard Hem Toftevaag, and Gabriele Meoni. torchquad: Numerical Integration in Arbitrary Dimensions with PyTorch. *J. Open Source Softw.*, 6(64):3439, 2021.
- [10] Ronald Kleiss and Roberto Pittau. Weight optimization in multichannel Monte Carlo. *Comput. Phys. Commun.*, 83:141–146, 1994.
- [11] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5), oct 2019.

<sup>1</sup><https://github.com/qiboteam/QiNNtegrate>.

- [12] Enrico Bothmann, Timo Janßen, Max Knobbe, Tobias Schmale, and Steffen Schumann. Exploring phase space with Neural Importance Sampling. *SciPost Phys.*, 8(4):069, 2020.
- [13] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. AutoInt: Automatic integration for fast neural volume rendering. *CoRR*, abs/2012.01714, 2020.
- [14] D. Maître and R. Santos-Mateos. Multi-variable integration with a neural network. *Journal of High Energy Physics*, 2023(3), mar 2023.
- [15] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, oct 2014.
- [16] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, sep 2017.
- [17] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii. Quantum circuit learning. *Physical Review A*, 98(3), sep 2018.
- [18] 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, 2020.
- [19] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, jun 2021.
- [20] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), mar 2019.
- [21] Gavin E. Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition, 2019.
- [22] Andrea Mari, Thomas R. Bromley, and Nathan Killoran. Estimating the gradient and higher-order derivatives on quantum hardware. *Physical Review A*, 103(1), jan 2021.
- [23] David Wierichs, Josh Izaac, Cody Wang, and Cedric Yen-Yu Lin. General parameter-shift rules for quantum gradients. *Quantum*, 6:677, mar 2022.
- [24] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adrián Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology*, 7(1):015018, dec 2021.
- [25] Stavros Efthymiou, Marco Lazzarin, Andrea Pasquale, and Stefano Carrazza. Quantum simulation with just-in-time compilation. *Quantum*, 6:814, sep 2022.
- [26] S. Carrazza, S. Efthymiou, M. Lazzarin, and A. Pasquale. An open-source modular framework for quantum computing. *Journal of Physics: Conference Series*, 2438(1):012148, feb 2023.
- [27] Andrea Pasquale, Stavros Efthymiou, Sergi Ramos-Calderer, Jadwiga Wilkens, Ingo Roth, and Stefano Carrazza. Towards an open-source framework to perform quantum calibration and characterization, 2023.
- [28] Stavros Efthymiou et al. qiboteam/qibo: Qibo 0.1.12, March 2023.
- [29] Stavros Efthymiou et al. qiboteam/qibolab: Qibolab 0.0.2, March 2023.
- [30] Andrea Pasquale et al. qiboteam/qibocal: Qibocal 0.0.1, February 2023.
- [31] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, aug 2018.
- [32] Andrea Delgado, Kathleen E. Hamilton, Prasanna Date, Jean-Roch Vlimant, Duarte Magano, Yasser Omar, Pedrame Bargassa, Anthony Francis, Alessio Gianelle, Lorenzo Sestini, Donatella Lucchesi, Davide Zuliani, Davide Nicotra, Jacco de Vries, Dominica Dibenedetto, Miriam Lucio Martinez, Eduardo Rodrigues, Carlos Vazquez Sierra, Sofia Vallecorsa, Jesse Thaler, Carlos Bravo-Prieto, su Yeon Chang, Jeffrey Lazar, Carlos A. Argüelles, and Jorge J. Martinez de Lejarza. Quantum computing for data analysis in high energy physics, 2022.

- [33] Gösta Gustafson, Stefan Prestel, Michael Spannowsky, and Simon Williams. Collider events on a quantum computer. *JHEP*, 11:035, 2022.
- [34] Gabriele Agliardi, Michele Grossi, Mathieu Pellen, and Enrico Prati. Quantum integration of elementary particle processes. *Phys. Lett. B*, 832:137228, 2022.
- [35] Christian W. Bauer et al. Quantum Simulation for High-Energy Physics. *PRX Quantum*, 4(2):027001, 2023.
- [36] Kinga Anna Woźniak, Vasilis Belis, Ema Puljak, Panagiotis Barkoutsos, Günther Dissertori, Michele Grossi, Maurizio Pierini, Florentin Reiter, Ivano Tavernelli, and Sofia Vallecorsa. Quantum anomaly detection in the latent space of proton collision events at the lhc, 2023.
- [37] Herschel A. Chawdhry and Mathieu Pellen. Quantum simulation of colour in perturbative quantum chromodynamics. 3 2023.
- [38] Matteo Robbiati, Juan M. Cruz-Martinez, and Stefano Carrazza. Determining probability density functions with adiabatic quantum computing, 2023.
- [39] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, nov 2019.
- [40] M. Cerezo, Andrew Arrasmith, Ryan Babush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms, 2020. cite arxiv:2012.09265Comment: Review Article. 29 pages, 6 figures.
- [41] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning, 2020.
- [42] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, feb 2020.
- [43] Massimiliano Incudini, Francesco Martini, and Alessandra Di Pierro. Structure learning of quantum embeddings, 2022.
- [44] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [45] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [46] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.
- [47] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, jan 2015.
- [48] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.
- [49] Matteo Robbiati, Stavros Efthymiou, Andrea Pasquale, and Stefano Carrazza. A quantum analytical adam descent through parameter shift rule using qibo, 2022.
- [50] Nikolaus Hansen. The cma evolution strategy: A tutorial, 2023.
- [51] Darrall Henderson, Sheldon Jacobson, and Alan Johnson. *The Theory and Practice of Simulated Annealing*, pages 287–319. 04 2006.
- [52] Jonas M. Kübler, Andrew Arrasmith, Lukasz Cincio, and Patrick J. Coles. An adaptive optimizer for measurement-frugal variational algorithms. *Quantum*, 4:263, may 2020.
- [53] Andrew Arrasmith, Lukasz Cincio, Rolando D. Somma, and Patrick J. Coles. Operator sampling for shot-frugal optimization in variational algorithms, 2020.
- [54] Matt Menickelly, Yunsoo Ha, and Matthew Otten. Latency considerations for stochastic optimizers in variational quantum algorithms. *Quantum*, 7:949, mar 2023.
- [55] James Stokes, Josh Izaac, Nathan Killoran, and Giuseppe Carleo. Quantum natural gradient. *Quantum*, 4:269, may 2020.
- [56] Jadwiga Wilkens. Quantum Circuit Library, 1 2023.
- [57] Adrián Pérez-Salinas, Juan Cruz-Martinez, Abdulla A. Alhajri, and Stefano Carrazza.

- Determining the proton content with a quantum computer. *Physical Review D*, 103(3), feb 2021.
- [58] Leonardo Banchi and Gavin E. Crooks. Measuring analytic gradients of general quantum evolution with the stochastic parameter shift rule. *Quantum*, 5:386, jan 2021.
- [59] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.*, 7:155–162, 1964.
- [60] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45, aug 1989.
- [61] David J. Wales and Jonathan P. K. Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, jul 1997.
- [62] Richard D. Ball et al. The path to proton structure at 1% accuracy. *Eur. Phys. J. C*, 82(5):428, 2022.
- [63] Richard D. Ball et al. An open-source machine learning framework for global analyses of parton distributions. *Eur. Phys. J. C*, 81(10):958, 2021.
- [64] Paolo Solinas, Simone Caletti, and Giovanni Minuto. Quantum gradient evaluation through quantum non-demolition measurements. *Eur. Phys. J. D*, 77(5):76, 2023.
- [65] Rodolfo Carobene, Alessandro Candido, Javier Serrano, Stefano Carrazza, and Edoardo-Pedicillo. qiboteam/qibosoq: Qibosoq 0.0.3, July 2023.
- [66] Leandro Stefanazzi, Ken Treptow, Neal Wilcer, Chris Stoughton, Salvatore Montella, Collin Bradford, Gustavo Cancelo, Shefali Saxena, Horacio Arnaldi, Sara Sussman, Andrew Houck, Ankur Agrawal, Helin Zhang, Chunyang Ding, and David I Schuster. The qick (quantum instrumentation control kit): Readout and control for qubits and detectors, 2022.