Do Quantum Models Make Good Generative Learners?

Kaitlin Gili

"Everything is teaching and learning, or at least, that's the best model I have come up with so far."

A thesis presented for the degree of
Doctor of Philosophy in Physics

Atomic and Laser Physics
University of Oxford
United Kingdom
November 2, 2023

Contents

1 PREFACE

This thesis is meant to be a summary of my research over the last three years; not a summary of my PhD. The latter is impossible to capture in this one text. It would require me to go way beyond the scope of Quantum Machine Learning (QML), and talk about *self-learning*, both the investigative process and resulting outcomes of discovering my personal values. It would require me to provide pictures of the 100+ beds I slept in while nomad-ing across 15 countries. I would want to share *what I learned* at each iteration step; the words documented in 20 handwritten journals that I dedicated to the closest people in my life. And perhaps the biggest thing that couldn't be expressed is the amount of gratitude that I have felt, and continue to feel, for the privilege of *learning* and *teaching* in every environment I became a part of. Overall, the QML research conducted was only a part of the PhD, as I believe it should be - although I never like the use of *"shoulds"* - and it is here that I combine these findings, that currently exist as several publications with my co-authors, into one body of work. As for the insights gained from the other parts of my PhD, that remains a work in progress for now.

2 INTRODUCTION

Around 2019, the Quantum Computing (QC) community experienced the *"quantum advantage"* boom, where higher quality QC devices were becoming more accessible to the public with a promise for solving problems that would be intractable for classical computers [1]. This rapidly became a unified goal for the Quantum Algorithms community, where applications in chemistry [2, 3, 4, 5, 6], quantum simulation [7, 8, 9], and machine learning (ML) [10, 11, 12] were identified as key targets for near-term advantage.

As such, quantum machine learning (QML) originated as an application for quantum computers, and has only recently been recognized as a separate field, combining elements from sub-areas as shown in Figure 1, that aims to understand how to design and evaluate quantum models for learning tasks. The goal of the field is still to build models that can rival or enhance classical networks in some meaningful way; however, the approach to achieving this goal is now more diverse among researchers.

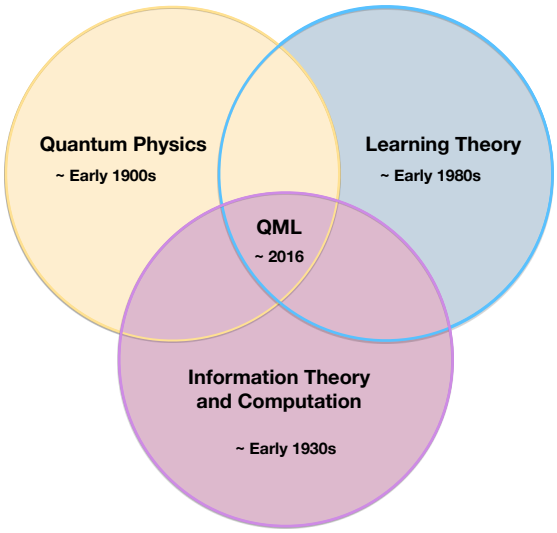


Fig. 1: **Representation of the field of Quantum Machine Learning (QML) as a combination of other sub-areas.**

The PhD research contributions presented here focus on enhancing the scientific community's understanding of how quantum computers could be useful for generative artificial intelligence (AI). These are powerful algorithms that can identify patterns among datasets and generate new data with the same underlying feature pattern. We have seen useful applications in classical generative AI across various fields such as molecular design and discovery [13, 14], image synthesis and deepfakes

generation [15, 16], risk communication for malware defense [17], and secure modeling of private data across industries [18]. With the development of quantum-inspired methods such as Tensor Network Born Machines (TNBMs) [19] and quantum models such as Quantum Circuit Born Machines (QCBMs) [20], generative models have been identified as leading candidates for quantum advantage applications. There have been several contributions that study the potential benefits and limitations of using quantum generative models as alternative or enhancers to classical models [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]. These insights, along with the contributions in this thesis, aim to answer a very exciting broad question:

Do quantum models make good generative learners?

Throughout this thesis, we will construct a "meta-model" of sorts as we attempt to answer this question with structured evidence. In the next two chapters, Chapter 3 and Chapter 4, we will provide an introduction to classical and quantum generative learning. These sections are meant to give technical background and present unified definitions for the most commonly referenced concepts throughout this thesis (e.g. what is a generative learner, an inductive bias, or a quantum neuron?). In Chapter 5, Chapter 6, and Chapter 7, we will begin to answer the broad question with results selected from the manuscripts and publications submitted during the PhD [32, 33, 34, 35][1]. More specifically, in Chapter 5, we will put forth a novel definition and evaluation method for determining a *good* generative learner. This framework for assessing quantum advantage with respect to generative modeling will then set the stage for Chapter 6, where we will use this method to assess one of the most promising quantum generative models in the literature. Following this, in 7, we will discuss intentional designs for quantum generative models, and put forth a novel hardware-aware and classically-inspired quantum architecture. We will demonstrate with numerical evidence and initial theoretical results that this model is a promising generative learner, and that this model can also be utilized to evaluate near term quantum hardware. To conclude, in Chapter 8, we claim that while the evidence presented here indicates a positive answer to our broad research question, there are future investigations to conduct such that we can obtain stronger support. We will end with a classification of the current research approaches within the QML community for generative learning, highlighting their relevance and

[1] Even though I am the primary author on these works; the results presented in this thesis would not be possible without the contributions of my collaborators. Specific contributors can be found in the Acknowledgements.

challenges.

3 WHAT IS GENERATIVE LEARNING?

In this chapter, we will breakdown generative learning into three main components: the data, the model, and the cost-based optimization procedure. Using these general components, we will discuss specific types of generative learners that have been proposed - specifically classical or quantum-inspired ones. The section will end with a review of how these models have been traditionally evaluated in the literature for their generalization performance. We will discuss theoretical and application-based approaches, which are important for the subsequent chapters that explore evaluation schemes for quantum generative learners relative to their classical counterparts.

3.1 The Learning Process

3.1.1 Definition

The general learning process consists of an internal model updating its representation of external data through an iterative feedback loop of guidance. As shown in Figure 2, the three necessary components are data, the model architecture, and the objective function [2] with an optimizer. Each element contains assumptions and biases that influence the overall learning process. Typically, we take the *inductive bias* of a model to be any intentional bias in the model architecture or training optimization scheme that encourages the model to more efficiently approximate the data. For example, an architecture that is attempting to learn movie preferences based off of the co-occurrence of features within previously watched media could start with a bias of another human's features who is your similar age or in your local area. Correct inductive biases can make training more efficient, whereas incorrect assumptions can require *even more* data to get the archiecture back on track.

3.1.2 Data

Data can be anything from the color of a t-shirt, to the name of a book purchased on Amazon, to the price increase of a pumpkin spice latte near Halloween. Data is simply *information*, and a dataset that can used in ML, is simply *a large amount of structured information*. To

[2] Throughout this thesis, we will use the words "objective", "cost", and "loss" function interchangeably.

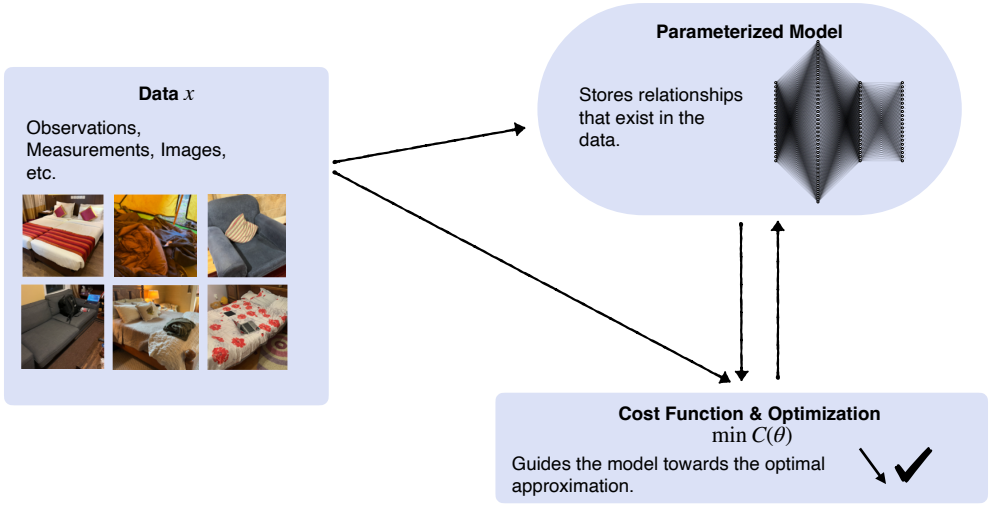


Fig. 2: **Visual diagram of the learning process: the feedback loop between three components such that the model is guided towards storing the optimal relationships with a given dataset.**

make it clear, *structured information* indicates that the dataset contains underlying patterns among its samples; we are working with datapoints that contain mutual information. Whether the dataset is labelled or unlabeled determines the *classification type* of the learning process - either supervised or unsupervised [3]. An example of an unsupervised training set is shown in Figure 2, where each picture does not contain a label of "bed" or "couch" like it would in a supervised setting [4]. Most of the data that is collected in the world *does not* come with a label assignment, which makes unsupervised learning a useful process for general pattern recognition.

Also, whether the dataset is discrete or continuous influences the learning process. The difference is that discrete datasets are composed of *countable* units (e.g. number of objects, individual molecules, etc.), while continuous datasets are composed of *measurable* units (e.g. temperature, time, etc.) in a spectrum. From here on out, we will focus on discrete datasets in the form: $\mathcal{D}_{\text{Data}} = \{x_1, x_2, ..., x_K\}$, where each sample x_i is an N-dimensional binary vector such that $x_i \in \{0, 1\}^N$ with $i = 1, 2, \ldots, K$. Each datapoint is modeled as an i.i.d sample from a probability data distribution $P_{\text{Target}}(x)$. A sub-set of data belonging to this set is known as the *training set*, which we can denote as $\mathcal{D}_{\text{Train}} = \{x_1, x_2, ..., x_T\}$. Some ML algorithms utilize an additional subset of data vectors as a test or validation set for performance checking at the end of the learning process.

[3] One can also have semi-supervised learning where only a portion of the dataset contains labels.
[4] These images are a part of a dataset collected of the 100+ places I slept during my nomad journey.

Lastly, data can be classified depending on the system it is coming from, e.g. classical or quantum data. For the remainder of this thesis, we can make this distinction in the following way: quantum data is generated from a quantum system, governed by the theory of quantum mechanics [36]; classical data is simply everything else. It is possible for some quantum data to be represented classically, and to also map classical data to a quantum Hilbert space, which often times makes the distinction between the decision-boundary of what is considered *classical* or *quantum* blurry. If the data from a quantum device can be represented classically efficiently, while it is still quantum data, it may not be data that is *most useful* to be processed on a quantum processor. As in, perhaps using classical computation is still more advantageous. Further discussion on quantum advantage will be provided in Chapter 5.

3.1.3 Models

A model is a structured representation of information. Examples of such representations are fully connected graph networks, Bayesian-decision trees, and feed-forward layers of artificial neurons. Each model type contains different building blocks as nodes to store parameter values, as well as a varied structure of activation potentials plus connectivities between nodes to exchange information. Examples of this are shown in Section 3.2.2. The information holders in the structure that can be tuned in the learning process are known as the *model parameters*, and this number determines the amount of *expressivity* that the model has to represent the data. Under-parameterizing does not provide the model with enough resources to represent the data; whereas over-parameterizing can make it more challenging for the optimal representation to be found during the training procedure.

As an example, let's consider a feed-forward Bayesian network. In this model, each parent node is only connected to its children's nodes, allowing all nodes in the same layer to be independent of one another. One can visualize it as a parameterized binary decision-tree with subsequent generational layers. Once trained on a dataset, the model represents a joint probability distribution $P_{\text{Model}}(x)$ over the nodes in the output layer, such that when sampling the model, one obtains a generated datapoint $\mathcal{D}_{\text{Gen}} = \{x_1, x_2, ..., x_G\}$, where each x_g is an N-dimensional binary bitstring, with $g = 1, 2, \ldots, G$, and $x_g \in P_{\text{Model}}(x) \approx P_{\text{Target}}(x)$. This method of obtaining data from a trained model is known as *sampling*, and the

computational hardness [5] of the sampling method depends on the model type as well. Note that the decision-tree structure of a Bayesian network is an example of a model having a specific *inductive bias* towards learning a dataset containing that particular structure. If the inductive bias *does* match the data, it can be extremely useful for narrowing the parameter space towards an optimal solution. Otherwise, an incorrect inductive bias can lead the model down a significantly poor parameter path. In Section , we will discuss specific examples of generative models, outlining their structure and sampling capabilities.

3.1.4 Objective Function & Optimizer

One can think of an objective function as a validation checker for the model, where it provides an optimizer with information, such that both guide the model in the optimal direction of the data. We call this optimization scheme, *training*, where we typically observe the number of iterations required for the model to minimize the difference between a representation of the training set and a representation of the model samples. The objective function $C(x) = \text{Diff}(P_{\text{Train}}(x), P_{\text{Model}}(x))$ may be computed at each step for the training evaluation, and the analytical or approximated gradient is typically used in a gradient-descent update rule for the model's parameters: $\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \nabla C(x)$, where $\boldsymbol{\theta}'$ is a new vector set of individual parameters θ_i for the next training iteration, Θ is the vector set from the previous iteration, and η is the learning rate that controls the size of the overall parameter shift in the direction defined by the gradient. Other hyper-parameters, variable controls in the optimization, can be defined and exploited to tune the optimization.

Different objective functions $C(x)$ exist with varying degrees of computational hardness when the dimension size of the problem increases. Additionally, there are optimizers with more or less degrees of freedom, which allows for hyper-parameter tuning beyond just that of adjusting the learning rate. These hyper-parameters influence the model's *trainability* - i.e. the model's ability to find a global optimium in the parameter space or a "good enough" local minimum. As objective functions and optimziers are typically discussed in the context of specific learning algorithms, specific examples for generative models will be discussed in Section 3.2.

3.2 Generative Learners

[5] We will always use computational hardness to indicate an exponential number of resources required to do a computation.

3.2.1 Definition

A more specialized case of the learning process is the *unsupervised generative learning process.* In this process, a model aims at capturing implicit correlations among unlabeled training data in order to generate samples with the same underlying features [37]. In other words, a generative model is trained on a limited set of data to learn optimal parameters that make the model a faithful approximation of the original target distribution $P_{\text{Target}}(x)$, implying that the model has the ability to generate data x_q from both inside and outside of the training set that are distributed according to this underlying distribution.

Nowadays, we can see large-scale classical generative models being used for challenging tasks such as recommendation systems [**?**], drug discovery [38], and image generation [39]. Models that vary in structure, unsurprisingly vary in functionality. Below, details of generative learners that will later be referenced in the research results of this thesis, are described in more detail.

3.2.2 Feed-forward Neural Networks

The basic building block of an Artificial Neural Network (ANN) is a neuron, containing parameter values θ_i^k that are either fixed or adaptable throughout training (visual representation in Figure 3). In a feed-forward network, neuron j in layer l_k is connected to every node in layer l_{k-1}; however, neurons in the same layer l_k are not connected. As for the optimized parameters θ_i^k: w_{ji}^k is the weight between neuron j in layer l_k and neuron i in layer l_{k-1}, and b_i^k is the bias for neuron i in layer l_k. Activation functions exist between neurons in the previous layer and the output neuron in the next layer such that the output neuron's parameters θ_i^k are a non-linear function of the previous layer's inputs:

$$\theta_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k o_j^{k-1}, \tag{1}$$

where θ_i^k is the total output value of neuron i in layer l_k prior to the activation function, o_j^{k-1} denotes the value for neuron j in layer l_{k-1}, r_{k-1} is the number of nodes in layer $k-1$, b_i^k is the bias for neuron i in layer l_k, and w_{ji}^k is the weight for neuron j in layer l_{k-1} for the incoming neuron i.

After this computation, an activation function $q(\theta_i^k)$ is performed to provide a value for the output neuron i in layer l_k.

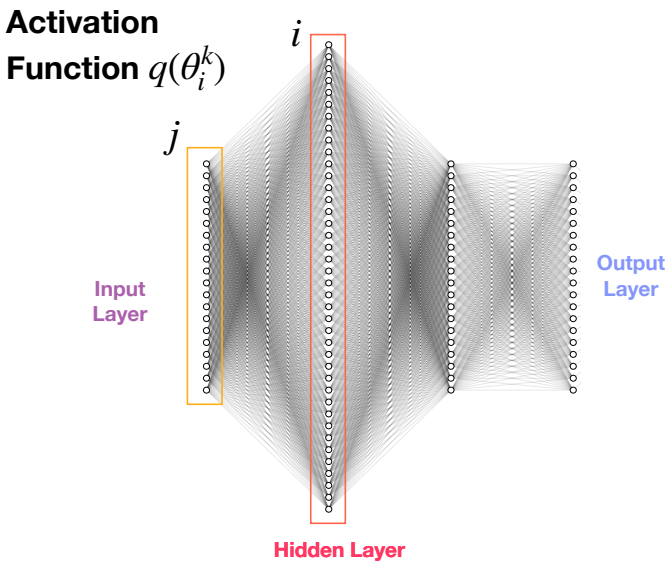


Fig. 3: **Visual diagram of a feed-forward neural network: each neuron i in layer l_{k-1} is connected to a single neuron j in layer l_k prior to an activation function on a the output θ_i^k .**

3.2.3 Generative Adversarial Network (GAN)

The Generative Adversarial Network (GAN) is a model architecture with a normal prior distribution and an adversarial training scheme, as described in the literature [40, 41, 37]. GANs are trained as two neural networks, a discriminator D and a generator G, competing against one another for optimal performance in an adversarial game. Samples from a prior distribution $q(z)$ are fed into the generator's input layer, and throughout training the generator attempts to produce new data x that can fool the discriminator into classifying x as a real rather than an artificially created data point. The goal of training is to maximize the generator's score and minimize the discriminator's score as described by the loss function:

$$\mathcal{L}_{\text{GAN}} = \min_G \max_D [\mathbf{E}_{x \sim P_{\text{Train}}}(x)[\log D(x)] + \mathbf{E}_{z \sim q(z)}[\log(1 - D(G(z)))]]. \tag{2}$$

For both the generator and the discriminator utilized later in this work, a feed forward architecture, as described in 3.2.2, is utilized with fully connected linear layers.

While GAN architectures have demonstrated great promise for image generation tasks [39], they typically fall short when it comes to generating *diverse* samples and are prone to mode-collapse behavior when training. This behavior is the result of the generator learning a single sample that

fools the discriminator every time, such that it gets stuck on that sample and does not learn anything new [37].

3.2.4 Tensor Network Born Machine (TNBM)

A Tensor Network Born Machine (TNBM), is a quantum-inspired generative model whose underlying architecture is chosen to be a Matrix Product State (MPS), a well-known 1D tensor network characterized by a low level of entanglement [19]. A TNBM takes unlabelled N-dimensional training bitstrings from the dataset $\{x_t\}_{t=1}^{T}$, and aims to encode the underlying probability distribution in a quantum wavefunction $|\psi\rangle$, expressing the correlations between samples in the amplitude of a quantum state, namely:

$$|\psi\rangle = \sum_{\{s\}} \sum_{\{\alpha\}} A^{s_1}_{\alpha_1} A^{s_2}_{\alpha_1\alpha_2} ... A^{s_N}_{\alpha_N} |s_1 s_2 ... s_N\rangle. \tag{3}$$

To motivate this representation, we note that an N-dimensional bitstring can be interpreted as a possible realization of the spin state (0,1) of N particles $|s_1 s_2 ... s_N\rangle$, and therefore the full quantum state can be written as a superposition of all the possible spin states. Rather than using the exact coefficient matrix to build $|\psi\rangle$, it can be approximated by the product of smaller parametrized single-particle matrices A^{s_i}, where the dimensions $\{\alpha\}$ are known as bond dimensions. The summation across α determines the probability amplitude for each superposition state of individual sites; thus, the bond dimensions controls the expressivity of the TNBM.

The training method is further described in Ref. [19], where models are trained via a DMRG-like algorithm with the log-likelihood cost function:

$$\mathcal{L}(\theta) = -\frac{1}{T} \sum_{t} \log(p_\theta(x_t)). \tag{4}$$

During training, samples are generated from the wavefunction according to the Born Rule:

$$p_\theta(x_t) = |\langle x_t|\psi\rangle|^2, \tag{5}$$

and the goal of the learning process is to find an optimal TNBM parametrization θ such that $p_\theta(x_t) \to P_{\text{Train}}(x_t)$.

A TNBM is known as a quantum-inspired technique as it builds upon fundamental concepts and formalism of the quantum-mechanical theory, but it is executed entirely on a classical platform.

3.3 Evaluating Models

Generative models are powerful and widespread algorithms, but the evaluation of their performance, especially on real-world datasets, is an open challenge. A huge variety of metrics and studies have been proposed to evaluate generative models, which can be found in two distinct sub-fields of machine learning (ML) research: computational learning theory [42, 43, 44] and models' performance benchmarking [45, 46, 18, 47, 38]. Here, we give a brief overview of these two areas of research, and to draw a clear distinction between them, we point to the advantages and challenges of each for evaluating unsupervised generative models. Subsequently, we focus on providing an overview of the main evaluation strategies that exist in this literature domain, pointing to Ref. [46, 46] for a thorough review.

3.3.1 Measuring Generalization

The language utilized in the sub-fields of computational learning theory and models' performance benchmarking varies greatly when discussing the evaluation approaches of unsupervised learning algorithms. There is a common goal of finding the best model (i.e., the one that 'generalizes' best); however, the optimal criterion and the generalization definition differ in the two perspectives.

In the context of computational learning theory, the optimal model is the one that has best approximately learned the ground truth probability distribution from the available training data [48]. Thus, generalization coincides with good inference capability. Upon taking this to be the definition of generalization, the model is able to achieve high-quality performance if its output distribution post-training is sufficiently close to the (unknown) ground truth. By using the Probably Approximately Correct (PAC) approach [48], one can derive worst-case generalization error bounds for a very broad range of models. These insights are incredibly useful for identifying clear cases in which models will not provide value, especially in the search for circumstances where quantum algorithms might exhibit an advantage over classical ones [30, 31, 44]. On real-world datasets, this definition of generalization can be extended to evaluating the difference between the trained model distribution and the empirical approximation of the ground truth, using a quantitative distance metric of choice.

However, this is where the definition of generalization in the context of computational learning theory diverges from that of the models'

performance benchmarking domain. For many practical problems, indeed, the optimal generative model is the one that can generate unseen high-quality data points that are solutions to a specific task, i.e., samples drawn from the ground truth distribution, but that did not exist in the empirical distribution used for training [49, 13, 38]. This implies that the emphasis is on the model being able to produce samples that come from the unseen part of the ground truth distribution: this capability of generating novel, diverse and good solutions is what is defined as generalization in this practical context [38]. Hence, if a model is provided with the complete set of solutions in the training process, it cannot generalize. Instead, since all the samples from the support of the ground truth distribution are given, the model would be restricted to exhibiting a behavior that we will later define as *memorization*, in even the best training scenario. In computational learning theory, this behavior would still be seen as a form of high-quality *generalization* performance, as long as the model learned the right features of the distribution. This is usually a case of interest in density estimation tasks; however, in many contexts, this behavior is distinct from practical generalization such that it can be detected when it is not useful for specific real-world applications, where the generative model is trained with the purpose of generating novel samples from the ground truth distribution.

In summary, the main difference between the two approaches is that in the models' performance benchmarking domain, the goal is to capture the model's generalization performance as a novel samples generator from the ground truth (*"efficient generator"*), not as a ground truth learning algorithm(*"efficient learner"*), as it is the case in computational learning theory. It is important to consider that an *"efficient learner"* does not always imply an *"efficient generator"* for a practical task at hand, and vice versa. The exact relation between the two approaches, especially its rigorous proof, is out of the scope of this thesis, but it is certainly an exciting avenue to bridge the gap between the two communities. The practical evaluation schemes, further described in Section 3.3.2, can augment the understanding of models' performance by providing a detailed picture, based on evaluating specific desired features of generated data, as well as by highlighting their tendency to exhibit training failures. However, this practical evaluation does not provide the same insights with regards to scaling complexity as those in computational learning theory. This is a regime where computational learning theory adds a large amount of value to our existing knowledge. Therefore, it should be

strongly emphasized that both research sub-fields are necessary to fully evaluate generative models, and that when possible, results from both realms should be included.

3.3.2 Alternative Metrics

A common approach to evaluate generative models uses statistical divergences, such as the Kullback-Leibler divergence [33] and the Total Variation Distance [31]. Unfortunately, the sample complexity of such quantities scales poorly with the dimensionality of the distribution under examination, proving them inadequate in high-dimensional spaces. To overcome this limitation, alternative evaluation metrics with polynomial sample complexity have been proposed, such as Inception Score (IS) [50], Frechét Inception Distance (FID) [51], and Kernel Inception Distance (KID) [52]. Additional strategies include utilizing kernel methods such as measuring the Maximum Mean Discrepancy (MMD) [43], or neural networks to estimate statistical divergences [53].

The main limitation affecting divergence-based metrics lies in that a single number summary is used to score a model, thus being unable to distinguish its different modes of failure. In light of this consideration, Ref. [54] introduced precision and recall as metrics to evaluate generative models, hence proposing a 2D evaluation to disentangle the various scenarios that can arise after training. Follow-up contributions have attempted to extend this idea from discrete to arbitrary probability distributions [55], and to improve precision and recall definitions and computation [56, 57].

This plethora of methods suggests how challenging it is to evaluate generative models. Evaluating the evaluation metrics themselves is an even more complicated task, despite the paramount importance of choosing the right metric for drawing the right conclusions [58]. Ref. [59] addresses such a problem, identifying a few necessary conditions that a metric should satisfy in order to qualify as a good performance estimator. One of these conditions is the ability of a metric to detect overfitting. As highlighted by Ref. [60], overfitting is basically equivalent to memorization, i.e., *anti-generalization*, and it is not always well defined, despite its importance.

While being well established in the context of image classification, notions of generalization are less standardized for generative models. Initial studies on this topic in the context of generative models can be found in Refs. [61, 53]. Nonetheless, none of the available metrics is specifically tailored to assessing generalization capabilities, or, in other words, to

detect overfitting upon occurrence [47]. So far, very few contributions have been proposed to address the interesting problem of studying and quantifying generalization from a real-world application perspective for generative models. This knowledge gap becomes exceedingly evident when looking at the recent literature contributions to the field of quantum generative modeling. Several of these works have hinted at the concept of generalization, but have ultimately restricted their results to replicating a given target probability distribution [19, 62, 63, 64, 49]. Leaving such a question for future research indicates the difficulty in benchmarking both classical and quantum models on real-world datasets for their generalization capabilities.

3.4 Chapter Summary

In this chapter, we explored a general definition for generative learning, as well as provided specific examples of generative learners, such as Artificial Neural Networks, Generative Adversarial Networks, Restricted Boltzmann Machines, and Matrix Product States. We introduced terminology that will be used throughout the rest of this thesis, as we transition to the discussion of quantum generative learners. We ended this chapter by discussing various approaches found predominantly in the classical ML literature for evaluating generative learners. In subsequent chapters, we will build upon these approaches to introduce a novel quantitative framework that can compare the performance of quantum and classical models on more equal ground.

4 WHERE DOES QUANTUM COMPUTING FIT-IN?

In this chapter, we will introduce *quantum computation*, and where this theory fits into the generative learning framework. We will focus on theoretical quantum models for *computing*, and then build upon these constructions to discuss quantum models for *learning*. Starting with a single bit of quantum information, we will build larger circuits that can be trained as generative learning algorithms.

4.1 Quantum Models for Computing

4.1.1 Quantum Bits

Quantum computers are composed of fundamental information units known as qubits [36]. Individual qubits are modeled using a 3D *Bloch Sphere*, where there are three parameters for an individual pure vector, or quantum state, that exists in some superposition of information states $|0\rangle$ and $|1\rangle$ represented as:

$$|\psi_{qubit}\rangle = \cos(\theta_1)|0\rangle + e^{-i\phi}\sin(\theta_2)|1\rangle \tag{6}$$

This state is the representation prior to measurement, where the outcome is interpreted as a $|0\rangle$ or $|1\rangle$ with some probability such that $|\cos(\theta_1)|^2 + |\sin(\theta_2)|^2 = 1$. Thus, we consider quantum hardware units to be *probabilistic* in comparison to our current transistor-based computing hardware. Rotations $R_X(\theta_1), R_Y(\theta_2), R_Z(\theta_3)$ around the sphere axes correspond to *single qubit gates*. These are modeled physical interactions that can change the state of the quantum system - i.e. manipulate the information in the system. These rotations are represented by unitary operations of a set of Pauli observables $\{\sigma_X, \sigma_Y, \sigma_Z\}$ such that $U(\boldsymbol{\theta}) = R_m(\boldsymbol{\theta}) = \exp\{(\frac{-i\theta\sigma_m}{2}\})$ for $m \in \{X, Y, Z\}$.

4.1.2 Quantum Circuits

Sets of qubits with logical operations, known as quantum circuits, in superposition and *entangled* with one another via multi-qubit gates, create large, general quantum states $|\psi\rangle$ that become difficult to store a classical representation of. If the system is in the general pure state $|\psi\rangle = a_i|a_i\rangle$, then we leverage a *general model* from quantum theory that the expectation value for some physical observable A to be measured on a quantum system is:

$$\langle\psi|A|\psi\rangle \tag{7}$$

Performing a measurement lands us in one of the eigenvalues of the system. Because the operators representing observables are self- adjoint, the spectral theorem for self adjoint operators allows us to write A as:

$$A = \sum_i a_i P_i^A \tag{8}$$

,

where a_i are the eigenvalues associated to the observable A and P_i^A are the projection operators onto the subspaces corresponding to those eigenvalues. We have that $P_i^A = |a_i\rangle\langle a_i|$. The probability of measuring A and getting the result a_i is given by:

$$P(a_i) = \langle\psi|P_i^A|\psi\rangle \tag{9}$$

In this kind of measurement, we completely collapse the quantum state to a probability of obtaining an eigenvector according to the observable in question - known as *Born's Rule*. This theory is extended to quantum computers, where large entangled quantum states are generated on quantum hardware (e.g. trapped-ions [65], superconducting materials [1], photons [66], etc.) and then measured in the computational basis (the Z eigenbasis as convention), outputting discrete data bitstrings x where $x_i \in \{0, 1\}\forall i$. The benefit of utilizing the quantum hardware is clear when we obtain large degrees of entanglement that become non-classically simulatable. Thus, one way in which quantum computers are predicted to be useful for problems where the amount of entanglement grows exponentially with the system size.

When the operator according to the observable is *parameterized* $A \rightarrow A(\boldsymbol{\theta})$, the matrix elements become a tunable variable for optimization. In the *variational* quantum circuit model [67], the observable is composed of many parameterized individual and multi-qubit gates acting on designated qubits. The structure in which these qubits are connected, or entangled, largely determines the *ansatz* of the model [6]. Other factors that influence the ansatz are the specific gate choices and the parameter initialization.

Algorithms of this nature became known as Noisy Intermediate Scale Quantum (NISQ) algorithms [68, 69], such that we could leverage current,

[6] The ansatz is essentially the initial starting point, or guess for the quantum state prior to the optimization. If the anstaz contains information of the dataset prior to optimization, this can be considered as an inductive bias that we discuss in classical ML models.

limited quantum resources to perform classically challenging tasks. The Parameterized Quantum Circuit (PQC) became the predominant computational model for a quantum computer that could be optimized with classical methods post-measurement to obtain solutions to interesting problems in quantum chemistry [2], finance [70], and ML [71].

Now one can make two connections to previous models. The first is that the TNBM, discussed in Section 3.2.4, is a reduced representation of a quantum state, where the operators do not contain complete information regarding the entanglement. There have been many works mapping TNBM's to quantum circuits, and even using them as ML models to find good initial inductive biases for quantum circuits [72]. Second, a parameterized single qubit is the simplest form of a variational quantum circuit.

4.2 Quantum Models for Generative Learning

4.2.1 Definition

PQC models can be used as the model component in the *learning process* discussed in Section 3.1.1, meaning that it can be trained to represent correlations within a dataset [71]. As shown previously that quantum circuits represent a probability distribution over a support of discrete bitstrings x, we now see that each quantum measurement in the computational basis is akin to generating a sample [71]. Thus, designing these parameterized circuits for generative modeling tasks is a natural idea. There is also theoretical evidence that generative models with certain quantum gate structures have more expressive power than classical networks [73]. However, the optimal design of these models for both NISQ and fault-tolerant hardware regimes is an open question. Some of the more prominent model architectures are further outlined below.

4.2.2 Quantum Circuit Born Machine (QCBM)

In the literature, QCBMs are one of the most popular quantum generative learners due to their highly expressive power [73] and the ability to perform direct sampling from the circuit as opposed to Restricted Boltzmann Machines (RBMs) that require a costly Gibbs sampling. This model family takes advantage of the Born rule of quantum mechanics to sample from a quantum state $|\psi\rangle$ learned via training of a Parameterized Quantum Circuit (PQC) unitary $U(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the vector of parameters for all of the single and entangling gates in the circuit. While alternative ansatz

connectivities may be implemented, we showcase a line topology in Figure 4 as a conventional choice for training circuits with very large depths. As the number of entangling gates scales linearly in the number of qubits for each layer, one can increase the number of layers with fewer number of parameters compared to other topologies. Note that alternative topologies, such as the all-to-all entangling connectivity available in ion-trap devices [20, 74, 75, 76], may be better for alternative tasks when the circuit depth can remain low. For a total number of layers L in the line circuit ansatz, each layer alternates between parameterized single-qubit rotation gate and multi-qubit entangling gate sequences until the final layer is reached. Each single-qubit gate sequence consists of an appropriate combination of Pauli X and Pauli Z rotations, $R_X(\theta)$ and $R_Z(\theta)$ respectively on each of the qubits, with $R_m(\theta) = \exp\{(\frac{-i\theta\sigma_m}{2}\})$. After each single-qubit gate layer, the entangling layer containing parameterized XX couplers between nearest neighbour qubits is executed, forming a line structure. Other topologies have also been explored (see e.g., Ref. [74]).

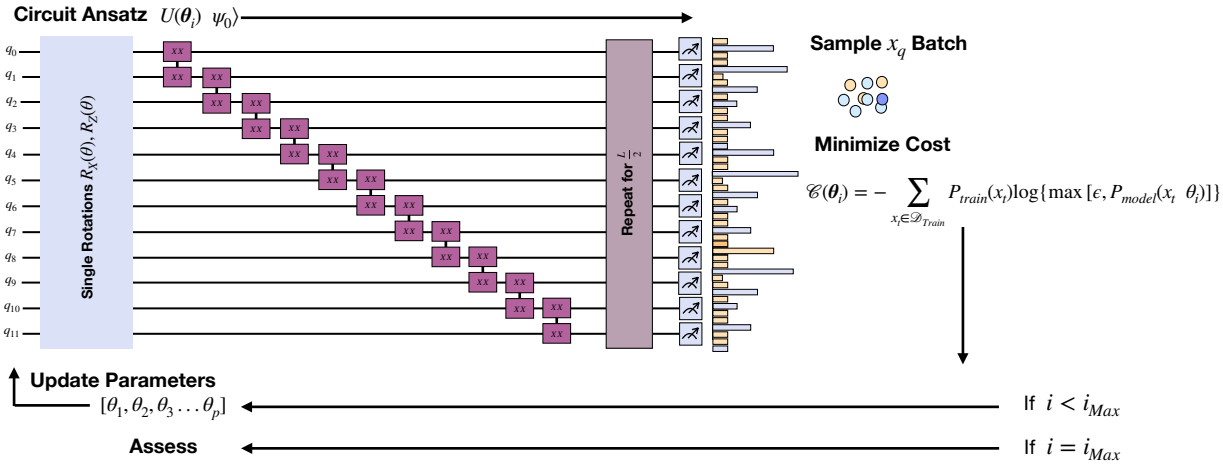


Fig. 4: **A visualization of the QCBM algorithm.** Starting with randomly initialized parameters, the 12-qubit circuit ansatz with a line topology of parameterized gates for an even number of layers L is shown (note that both single-qubit and entangling gates are taken to be parametrized). Measuring in the computational basis provides samples distributed according to the probabilities encoded in the quantum state $|\psi\rangle$ that results from performing the unitary operation $U(\boldsymbol{\theta})$ on an initial quantum state $|\psi_0\rangle$. Iterative training is implemented up to a number of i_{Max} iterations in order to optimize the circuit parameters via minimization of the cost function $\mathcal{C}(\boldsymbol{\theta}_i)$.

To minimize the number of variational parameters of the circuit and favor its trainability without sacrificing its expressive power, we can utilize the following strategy to carefully design the single-qubit layers [20], for an even L. In the first single-qubit gate sequence, we choose to decompose the arbitrary single-qubit transformation as $R_Z(\theta_1)\ R_X(\theta_2)\ R_Z(\theta_3)$. Since our initial state is $|00\cdots0>$, the first sequence of $R_Z(\theta_1)$ only adds a global

phase to the quantum state, which is irrelevant since it will get washed out once we consider the Born's probabilities. Therefore, we can remove this first sequence of $R_Z(\theta_1)$ on each qubit without reducing the circuit's expressive power. For the next single-qubit sequences after the very first one, we use a decomposition of the form $R_X(\theta_1)\ R_Z(\theta_2)\ R_X(\theta_3)$: it can be seen that the commutation of R_X with XX would lead to "collapsing" one sequence of R_X from the single-qubit sequence right before the entangling layer into the one right after it. Leveraging this commutation-and-collapse trick, all the single-qubit sequences after the first can also be reduced to $2N$ gates, except for the last one that has $3N$ gates as the final R_X doesn't have any following rotation to collapse into. For N qubits and an even L, this ansatz choice gives a total number of parameters $P = (3L/2+1)N-(L/2)$. When $L = 2$, the parameter count is simply given by $P = 3N - 1$ as there are only $2N$ single qubit gates in the first single-qubit sequence, as previously explained, followed by $N - 1$ parametrized XX gates.

During each training iteration i, up to i_{Max}, the quantum circuit is simulated or run on quantum hardware with the current iteration parameter values $\boldsymbol{\theta}_i$, and is then queried to generate samples x_q according to the following model distribution [20]:

$$P_{\text{Model}}(x_q|\theta_i) = |\langle x_q|\psi(\theta_i)\rangle|^2. \tag{10}$$

The generated samples are input into a classical cost function that measures the distance between the model output samples x_q and the training samples x_t taken from the underlying target distribution. Typical cost functions include the negative log likelihood (NLL) and the Maximum Mean Discrepancy (MMD) loss [26]. The NLL cost function at each iteration defined as:

$$\mathcal{C}(\boldsymbol{\theta}_i) = -\sum_{x_t \in \mathcal{D}_{Train}} P_{\text{Train}}(x_t) \log\{\max\left[\epsilon, P_{\text{Model}}(x_t|\theta_i)\right]\}, \tag{11}$$

where $\epsilon = 10^{-8}$ mitigates the singularity that occurs when $P_{\text{Model}}(x_t|\theta_i) = 0$.

Another version of this cost function is known as the Kullback-Leibler Divergence [77] defined as:

$$KL = \sum_x P_{\text{Target}}(x) \log\left(\frac{P_{\text{Target}}(x)}{\max(P_{\text{Model}}(x), \epsilon)}\right), \tag{12}$$

where $\epsilon \approx 10^{-16}$ such that the function remains defined for $P_{\text{Model}}(x) = 0$. When training the model, the desire is to obtain the following:

$$KL(P_{\text{Target}}(x), P_{\text{Model}}(x)) = 0 \tag{13}$$

This means that the model's distribution is identical to the target, and thus the model can fully express the target distribution. A limitation of these cost functions is that they require explicit access to $P_{\text{Model}}(x_q)$, where we are only able to approximate this value with a finite number of queries Q taken from the trained model. Thus, this cost function is prone to the curse of dimensionality; it becomes challenging to approximate when scaling to larger data dimensions. Once computed, the cost is utilized in a gradient-based or gradient-free optimization scheme that updates the parameter values and feeds them back into the circuit for the next iteration. After a specified number of iterations i_{Max} (once a cost-threshold α is reached or some other convergence criterion is satisfied), the trained model can be queried and its output is used for evaluation.

There have been several proposals to mitigate the singularities and scalability issues arising from the use of the NLL (or, equivalently, the KL Divergence). The first proposal to mitigate this was to use the Maximum Mean Discrepancy (MMD) cost function [26]. Other researchers have proposed other types of divergences, such as the Sinkhorn divergence and the Stein discrepancy [21], and other f-divergences [78]. Although changing the cost function might mostly help to reduce significantly the resources needed to estimate the cost function itself from the quantum device's samples, this might not suffice to address other trainability issues such as the presence of barren plateaus - i.e. vanishing gradients during training. This is precisely the regime where initialization ([72] and [79]) and constructing intentional ansazte (inductive biases) becomes a priority.

4.2.3 Quantum Neuron

As the basic building block of the classical feed-forward neural network, discussed in Section 3.2.2, is the *neuron*, it is only natural to consider a quantum version of this model. This is a more intentionally constructed model in contrast with the general PQC framework models presented above. There have been many proposed models for the *quantum neuron* [80, 81, 82], and finding the optimal analogue is still an open research question [11]. Here, we discuss the most relevant neuron proposal to this thesis, which was proposed by Ref [82]. In this proposed quantum neuron (QN) circuit, each neuron is assigned a qubit, so the QN circuit has an input register $|x_{in}\rangle$ representing the previous layer, an output qubit

initiated in the state $|\psi_{out}\rangle$ representing the neuron of the next layer, and also an ancilla qubit, initially in $|0_a\rangle$. A visual is displayed in Figure 19.

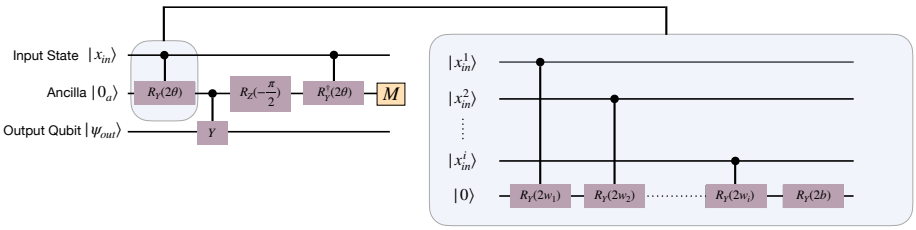


Fig. 5: **An example of a single quantum neuron circuit** Left: Quantum Neuron circuit, connecting neurons (qubits) from previous layer to a single neuron in the next layer. If the ancilla measurement outcome is 0, the circuit has successfully applied $R_Y(2q(\theta))$ to the output qubit, and if the measurement outcome is 1, one needs to apply $R_Y(-\pi/2)$ to the output qubit, a NOT gate to the ancilla, and apply the circuit again. Hence the Quantum Neuron circuit is a repeat-until-success circuit. Right: implementing $R_Y(2\theta)$, where θ is a function of weights, biases, and input neuron values.

Suppose initially that $|x_{in}\rangle$ is a single bitstring, rather than a superposition of bitstrings. Then, just before the measurement (see Figure 19 left), the combined quantum state is:

$$\begin{aligned}|\psi\rangle = |x_{in}\rangle \otimes (\sqrt{p(\theta)}\,|0_a\rangle \otimes R_Y(2q(\theta))\,|\psi_{out}\rangle \\ +\sqrt{1-p(\theta)}\,|1_a\rangle \otimes R_Y(\pi/2)\,|\psi_{out}\rangle),\end{aligned} \tag{14}$$

where θ is the weighted sum of neuron activations in the previous layer:

$$\theta = w_1x_1 + w_2x_2 + ... + w_nx_n + b, \tag{15}$$

$w_n \in (-1;1)$ are the weights and $b \in (-1;1)$ is the bias. Notice the argument of the y-rotation is not 2θ, but $2q(\theta)$: this is the activation function

$$q(\theta) = \arctan\big(\tan^2(\theta)\big). \tag{16}$$

The activation function has a sigmoid shape, and is determined by the sequence of quantum gates applied. In any case, by simply measuring the ancilla qubit to be $|0_a\rangle$ with a probability $p(\theta) > \frac{1}{2}$ [82], we end up in the right state for the next layer:

$$|\psi\rangle = |x_{in}\rangle \otimes |0_a\rangle \otimes R_Y(2q(\theta)\,|\psi_{out}\rangle\,. \tag{17}$$

Importantly, when probability $1 - p(\theta) >= \frac{1}{2}$ we will end up in the state:

$$|\psi\rangle = |x_{in}\rangle \otimes |1_a\rangle \otimes R_Y(\pi/2)\,|\psi_{out}\rangle\,, \tag{18}$$

which can be returned to the pre-circuit state with a NOT gate on the ancilla and $R_Y(-\pi/2)$ applied to the output qubit. The process will then be repeated until the ancilla measurement yields $|0_a\rangle$, thus the Quantum

Neuron circuit belongs to the class of Repeat Until Success (RUS) circuits [83, 84].

Importantly, if we allow the input register to be in a general state - as a *superposition* of bitstrings $\sum_i \left|x^i\right\rangle$, then the circuit will successfully apply the appropriate transformations to each of the bitstrings in the superposition, resulting in the final state:

$$\sum_i F_i \left|x_{in}^i\right\rangle \otimes |0_a\rangle \otimes R_Y(2q(\theta^i)) \left|\psi_{out}\right\rangle . \tag{19}$$

Here, F_i refers to an amplitude deformation in the input state during the RUS mapping. These QN building blocks have been previously used for designing small-scale quantum networks for supervised learning tasks [82]. In Section 6, we will extend off of this model to construct a novel quantum generative learner.

4.3 Chapter Summary

In this chapter, we discussed how quantum computers *fit* into the generative learning framework by introducing theoretical models for quantum computation and learning, previously defined in the literature. These models will be references in later research-focused chapters that investigate and improve upon these models.

5 WHAT IS A GOOD QUANTUM GENERATIVE LEARNER?

5.1 Background

The question *where does quantum fit-in?* is fundamentally different from asking *do quantum models make good generative learners?*. In order to address this question, we must first define the most unambiguous component - what we mean by *good*.

As quantum ML began as a potential *advantage application* for quantum computers, it has been natural for utility to mean that a generative learner provides some form of *quantum advantage* over a classical alternative.

Quantum advantage is generally intended as the capability of quantum computing devices to outperform classical computers, providing exponential speedups in solving a given task, which would otherwise be unsolvable, even using the best classical machine and algorithm [1, 85, 86, 68, 87, 88]. In recent years, a large part of the quantum computing community has been gravitating toward a more concrete definition of quantum advantage, namely *practical quantum advantage* (PQA), also propelled by the growing interest from technology firms and companies in various application domains. Practical quantum advantage indicates the quest for quantum machines that can solve problems *of practical interest* that are not tractable for traditional computers [8, 89]. In other words, practical quantum advantage is the ability of a quantum system to perform a useful task faster or better than is possible with any existing classical system [90]. As long as the superiority is demonstrated in the real-world setting, under the real constrains and problem size of interest, one can waive the need for demonstrating an asymptotic scaling with problem size, which is the usual emphasis in algorithmic quantum speedup [91].

In this chapter, we build off of this general definition to evaluate generative learning algorithms for quantum advantage - work that was published in [32]. In this framework, we suggest that generative models' performance be assessed by their capability to *generalize*, i.e., generate new high-scoring diverse solutions for the task of interest [38, 49]. We highlight that our proposed definition of generalization differs from the one outlined within the theoretical setting of computational learning theory [42, 30, 29, 31], i.e., a model's ability to learn the ground truth probability distribution given a limited set of training data. Our approach follows closely the definitions and frameworks used by other ML practitioners (see e.g., [45, 38]) discussed in Section 3.3.1, which focus on scalable and practical methods to evaluate the performance of generative

models.

To the best of our knowledge, this is the first proposal of an approach that combines a heuristic-based analysis with an application-based dataset to quantitatively evaluate generalization of unsupervised generative models and to directly compare classical and quantum-inspired models side by side in search for practical quantum advantage.

5.2 New Evaluation Framework

5.2.1 Definitions

Since the goal is to compare the generalization performance of models for measuring practical quantum advantage, here, we introduce formal definitions to quantify different aspects of the practical behaviours that arise when we sample from the generative model. To further distinguish these definitions from those in computational learning theory, we provide the contextual names: validity-based and quality-based generalization.

Pre-Generalization We define *pre-generalization* as the generative model's ability to go beyond the training set $\mathcal{D}_{\text{Train}}$ by producing unseen outputs. More precisely, for any level of generalization to occur it is necessary - but not sufficient - that there exist some points x_g such that

$$x_g \in \mathcal{D}_{\text{Gen}} \wedge x_g \notin \mathcal{D}_{\text{Train}}. \tag{20}$$

However, these outputs may not be samples distributed according to $P_{\text{Target}}(x)$; for example, they may just be meaningless noise instead. In other words, pre-generalization is the model's ability to generate any new output - whether it is distributed according to $P_{\text{Target}}(x)$ or not (shown in Figure 6). Note that this behaviour is a prerequisite for a model to be able to generalize, and not generalization in and of itself. As mentioned above and further specified below, to have any kind of generalization, a model must first be able to generate data beyond the training set, and the generalization potential is higher if the amount of unseen data is maximized. This implies that the training set cannot be exhaustive, i.e. the number of unique[7] training bitstrings must be less than the number of unique bitstrings that can be sampled from $P_{\text{Target}}(x)$. To discover new data, the training dataset should not consist of all of the bitstrings that could be sampled from the original distribution (i.e. its support).

[7] Bitstrings = {00, 00, 11}, unique bitstrings = {00, 11}.

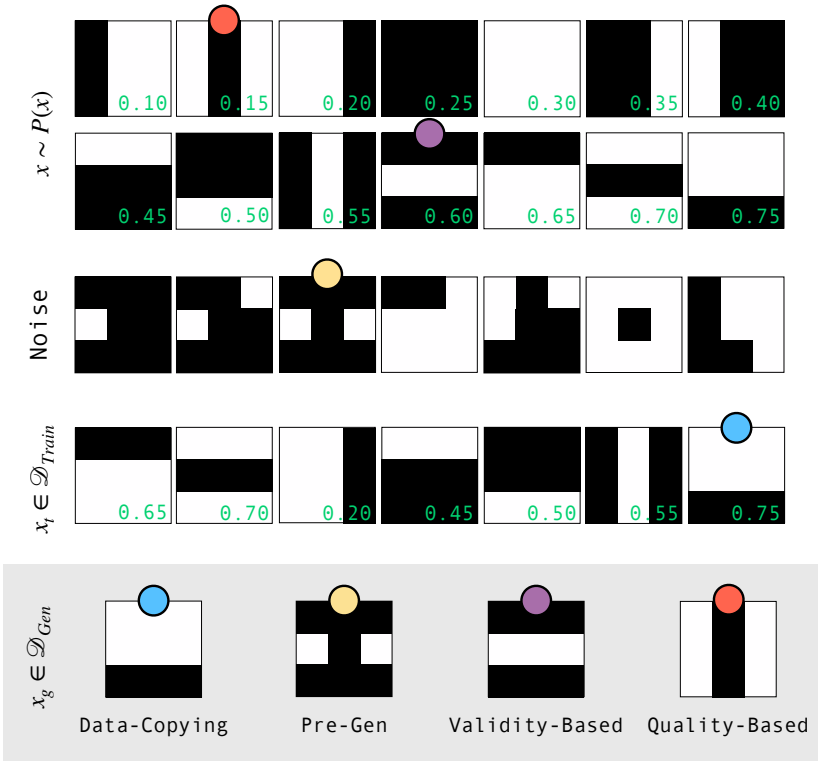


Fig. 6: **A visual representation of generalization-related concepts.** The figure shows the different behaviours a model can exhibit when generating data, using a 3x3 Bars and Stripes dataset as an example. The top two rows display a set of samples x distributed according to the data distribution $P_{\text{Target}}(x)$; note that only a subset of the 3x3 Bars and Stripes dataset is displayed, rather than the full set of patterns. The third row contains samples that do not belong to this dataset (Noise). The fourth row contains a subset of samples $x_t \in \mathcal{D}_{\text{Train}}$ used for training and distributed according to $P_{\text{Train}}(x_t)$, while the bottom row shows a new set of samples x_g produced by the model and living in $\mathcal{D}_{\text{Gen}}$. Note that each sample contains an associated toy-score that corresponds to the samples' associated cost. In this toy example, the samples are assigned a real-valued score in $(0, 1)$, except for noisy samples that don't have an associated cost as they are not part of the valid solution space. The bottom row displays four samples from the generated queries, each of which is tagged with a different model behaviour: memorizing data from $\mathcal{D}_{\text{Train}}$ (blue dot), producing data outside of $\mathcal{D}_{\text{Train}}$ that may be noise (yellow dot), generalizing to new data distributed according to $P_{\text{Target}}(x)$ (purple dot), and generalizing to new data distributed according to $P_{\text{Target}}(x)$ that contains a minimum value to an associated cost function (red dot).

The pre-generalization behaviour can be verified with the exploration metric E, defined in Section 5.2.3, that quantifies how many generated samples were not included in the training set. Note that this quantity has a similar definition to the *authenticity* metric in Ref. [18], that captures sample novelty. However, the exploration metric is computed directly from samples rather than requiring an embedding scheme and a separate classification network. This quantity allows one to investigate the general questions: *"Can the model reach out-of-training data points? And with which frequency?"*.

Validity-Based Generalization We define*validity-based generalization* as the generative model's ability to go beyond the training set $\mathcal{D}_{\text{Train}}$ and effectively produce new bitstrings living in a given solution space with the underlying distribution $P_{\text{Target}}(x)$ (shown in Figure 6). In other words, the model is able to learn a fixed particular feature about bitstrings drawn from $P_{\text{Target}}(x)$ and produce new samples with the same feature, where this feature is specified via a constraint on the bitstrings. More precisely, the generative model outputs samples x_g such that:

$$x_g \notin \mathcal{D}_{\text{Train}} \wedge x_g \in \text{support of } P_{\text{Target}}(x). \tag{21}$$

This approach for validity-based generalization is task-independent, as the metrics are exclusively sample-based and agnostic to the specific use case, or more specifically, independent of the quality associated to each bitstring. In Section 5.2.3 we highlight the essential conditions one needs to meet when defining an appropriate task to study validity-based generalization.

The validity-based generalization behaviour is evaluated with three metrics: *fidelity* F, *rate* R, and *coverage* C. In a nutshell, F quantifies the probability that a model generates unseen samples that are valid results rather than unwanted noise. R quantifies the frequency at which a model produces unseen and valid results. C quantifies the fraction of unseen and valid results retrieved among all the potential valid and unseen samples. These metrics allow one to answer the following general questions, respectively linked to the three generalization estimators presented above:

- F: "How effectively can the model distinguish between noisy and valid unseen results?"
- R: "How efficiently can the model reach unseen and valid results?"
- C: "How effectively can the model reach all unseen and valid results?"

Quality-Based Generalization We define *quality-based generalization* as the generative model's ability to go beyond the training set $\mathcal{D}_{\text{Train}}$ and effectively produce bitstrings living in a given solution space with underlying distribution $P_{\text{Target}}(x)$, where the new bitstrings can be mapped to a real number indicating their quality. While there can be many examples of functional maps that one could use to assign each bitstring a score to be maximized, we emphasize optimization as a natural choice for assigning such a value to each sample, as proposed by Refs. [49, 92, 38]. In this case, the score is quantified by a cost to be minimized. In other words,

optimization provides a natural framework to introduce quantitative estimators of generalization, as a generative task can be equipped with a well-defined cost function, indicating the quality of samples. The framework presented here combines generalization and optimization as a promising strategy towards the definition of quantitative metrics. We highlight that if one uses a generative model as an optimizer, the success of the algorithm depends on the generation of high-quality solution candidates, rather than inferring the ground truth data distribution as it is the case in computational learning theory.

When focusing on quality-based generalization, one is interested in generating samples that satisfy a validity criterion, but also have associated costs that minimize a given objective function (shown in Figure 6).

A generative model thus exhibits quality-based generalization if it is able to produce at least some unseen and valid samples that have on average similarly low (or lower) cost values than the ones associated to at least some of the training samples. More precisely,

$$x_g \text{ satisfies Eq. } 21 \wedge f(\mathcal{D}_{\text{Gen}}, c(x)) < f(\mathcal{D}_{\text{Train}}, c(x)), \tag{22}$$

for a given suitable function f (e.g., the minimum sample cost $c(x)$ in each sample set) that depends on how strict the cost minimization requirements are for the problem under examination.

Developing metrics for assessing quality-based generalization is a task-dependent challenge as it allows one to evaluate the model's *sample quality*, according to a specific task and measured by its associated cost function.

In Section 5.2.3, we introduce two versions of the sample quality metric, induced by a different choice of f: the first one evaluates the model's ability to generate a minimum cost value that is lower than anything in the training set, whereas the second accounts for a diversity of samples whose cost is below a user-defined percentile threshold. Even though the former could seem more adequate to quantify the generator's ability to go beyond the sample quality available in the training set, it may be the case that producing the lowest cost value is not the only desired behaviour of the task. For instance, it may be that the desired behaviour is to generate diversity of new samples with a cost comparable to the lowest values found in the training set. In this scenario, the latter version allows one to reward alternative solutions without restricting the model only toward values below the training threshold. Since for many practical optimization tasks one cares about reaching a diverse pool of high-quality solutions, we

also see value in considering the number of unique samples with a lower cost value than a user-defined threshold in the training set (e.g., the minimum value in the training set).

The quality-based generalization metrics allow us to investigate the general question: *"Can the model reach unseen and valid results that are more or just as valuable than the best in the training set?"*.

5.2.2 Requirements

In order to properly assess generalization from the practical perspective, the generative model's task must meet some essential requirements. Such assumptions do not limit the scope of our approach as they simply provide a robust definition of the task at hand.

As previously specified, we focus the analysis on binary encodings of discrete datasets $\mathcal{D}_{\text{Train}} = \{x_1, x_2, ..., x_T\}$, with $x_t \in \{0,1\}^N$. One can thus identify a search space $\mathcal{U}$ of size 2^N, that contains all possible N-dimensional bitstrings. For validity-based generalization, there must exist a subspace of $\mathcal{U}$ containing the set of bitstrings we would like our trained model to generate. We can refer to this as the valid solution space $\mathcal{S}$, that includes all the samples that exhibit a given desired feature. Hence, the model aims to approximate the underlying unknown *data distribution*, defined as:

$$P_{\text{Target}}(x) = \frac{1}{|\mathcal{S}|}, \forall x \in \mathcal{S}. \tag{23}$$

The notion of validity produces a non-trivial distribution of valid samples across the overall search space $\mathcal{U}$, adding complexity to the problem despite the data distribution being uniform over the solution space $\mathcal{S}$. we emphasize that this general solution space $\mathcal{S}$ will contain different bitstrings for various representational datasets of interest. For instance, Figure 6 displays samples from the well-known Bars and Stripes dataset [93]: in this case, the solution space $\mathcal{S}$ would contain all valid bar and stripe patterns, some of which are shown in the top row of the figure. Alternative datasets could focus on solution spaces defined by a parity constraint, by a cardinality constraint or by any other property of interest. The solution space must have a well defined notion of validity that can be evaluated for each of the bitstrings in $\mathcal{U}$ to verify whether or not they live in its subset $\mathcal{S}$.

The model's task is therefore to generate novel samples in $\mathcal{S}$, after a learning process involving a limited number T of unique training samples, i.e., $T = \epsilon|\mathcal{S}|$, where the seen portion $\epsilon \ll 1$ is a small parameter

quantifying the percentage of $\mathcal{S}$ that gets seen during training. Note that this is a necessary requirement for generalization because it guarantees that the training set is not exhaustive.

With T training samples, the model has access only to an approximated version of the data distribution, that we denote as the *training distribution*:

$$P_{\text{Train}}(x) = \frac{1}{T}, \forall x \in \mathcal{D}_{\text{Train}}. \tag{24}$$

For quality-based generalization, there is an additional requirement as this behaviour depends not only on the validity of the bitstrings, but also on the value associated to each pattern, according to a cost function $c(x)$. As such, in order to assess quality-based generalization, it is necessary for the task of interest to have a well-defined objective function that indicates the cost of each bitstring, in search for minimum values.

As the objective is for the model to learn the valid bitstring patterns as well as to generate patterns with low-cost values, it is integral to re-weight the dataset distribution in Equation 23. Here we use a *softmax* function in order to introduce cost-related information in the training data set. In this scenario, the training samples approximate the following *re-weighted training distribution*:

$$P^{(\text{w})}_{\text{Train}}(x) = \frac{e^{-\beta_m c(x)}}{\sum_{i=1}^{T} e^{-\beta_m c(x)}}, \forall x \in \mathcal{D}_{\text{Train}}. \tag{25}$$

Following Ref. [49], $\frac{1}{\beta_m}$ was chosen to be the standard deviation of the costs in the training data, whereas $c(x)$ is the cost of each sample bitstring.

In summary, the two main essentials for evaluating respectively validity-based and quality-based generalization are the following:

- There exists a well-defined solution space $\mathcal{S}$, containing bitstring patterns that are valid according to easy to specify and verify constraints.
- There exists a well-defined cost function $c(x)$ that can be computed to assess the generalization for all valid bitstring patterns.

5.2.3 Metrics

As described in Section 5.2.1 and Section 5.2.3, practical generalization occurs when a model generates novel samples that display desired features and belong to the support of some underlying distribution. To give a quantitative definition of the validity- and quality-based generalization

metrics, We first need to clarify the nomenclature of all the spaces involved. We have already defined the collection of all queries generated by a trained generative model as $\mathcal{D}_{\text{Gen}}$, where $|\mathcal{D}_{\text{Gen}}| = Q$. We can then call $\mathcal{G}_{\text{sol}}$ the multi-set of all valid and unseen queries, which reflect the model's validity-based generalization capability. Now, we can further define a subset of $\mathcal{G}_{\text{sol}}$ that contains all its unique bitstring solutions as g_{sol}, thus the only difference between $\mathcal{G}_{\text{sol}}$ and g_{sol} is that in the latter each bitstring appears only once, whereas in the former there can be many occurrences of the same sample. Lastly, we can define the multi-subset of unseen queries as $\mathcal{G}_{\text{new}}$, where some of these queries might be unwanted noise and hence reflect the model's exploration capability. Note that we will use uppercase variables for multi-sets and lowercase variables for unique sets, and a visual representation of the sets in play can be found in Figure 7.

Having clarified the nomenclature of the spaces involved in the task, we can now proceed to the definition of the generalization metrics.

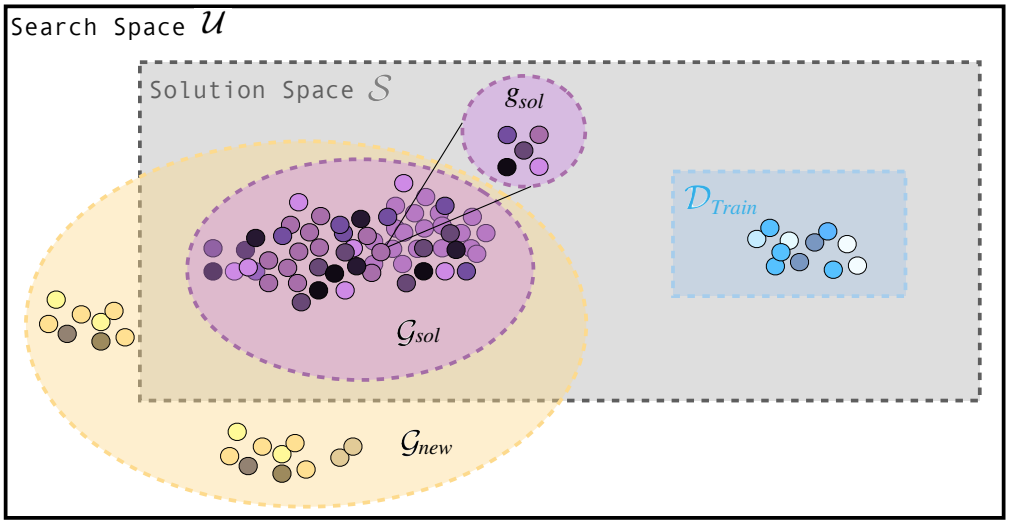


Fig. 7: **A visual representation of all possible spaces where a generated query might be located.** Each query is represented by a color-coded dot, where the color-code is the same as in Figure 6 (Data-Copying: blue, Pre-Generalization: yellow, Validity-Based Generalization: purple) and the color-shade represents a unique bitstring sample. One can take all non-unique queries outside of the training set to be in the multi-subset $\mathcal{G}_{\text{new}}$ (inside the yellow oval), whether they are in the solution space $\mathcal{S}$ or not. Furthermore, we take $\mathcal{G}_{\text{sol}}$ to be all non-unique queries that exist in the solution space (inside the pink oval) and g_{sol} to be all of the unique queries among $\mathcal{G}_{\text{sol}}$ (zoomed-in). Lastly, if a query exists in $\mathcal{D}_{\text{Train}}$, it is a memorized count from the training set. Note that the quality-based queries (not shown) must exist inside of the solution space.

Evaluating Pre-Generalization While a model's capability to generate unseen samples that are not valid or valuable solutions to the task at hand is not considered generalization behaviour in and of itself, it is an important prerequisite for generalization from the practical perspective. If the model

is not able to go beyond the training set, even just to produce noisy outputs, then the model is not passing the first requirement for generalization - the ability to produce novel data points. To conduct a pre-generalization evaluation prior to assessing for any kind of validity-based or quality-based generalization, we introduce the *exploration* metric E, that quantifies the fraction of generated queries that are new data points, namely:

$$E = \frac{|\mathcal{G}_{\text{new}}|}{Q}. \tag{26}$$

If $E \approx 0$, the model will not pass the first required check for practical generalization. This may be due to an intrinsic property of the model, i.e., the inability to generate novel data, or it can be an artifact of the training set being (almost) exhaustive, because nothing new can be generated if the training data covers (almost) all the entire valid space.

Evaluating Validity-Based Generalization Here, we introduce three sample-based metrics that describe each model's validity-based generalization behaviour after training: fidelity F, rate R, and coverage C.

Fidelity describes the model's ability to distinguish an unseen and valid sample in $\mathcal{S}$ from a meaningless output (i.e., noise) and it quantifies the fraction of unseen queries that fall into the unseen solution space. It is defined as follows:

$$F = \frac{|\mathcal{G}_{\text{sol}}|}{|\mathcal{G}_{\text{new}}|}. \tag{27}$$

Rate describes the model's ability to efficiently produce unseen and valid samples and it quantifies the fraction of all queries that fall into the unseen solution space, namely:

$$R = \frac{|\mathcal{G}_{\text{sol}}|}{Q}. \tag{28}$$

Coverage describes the model's ability to recover all unique unseen and valid samples and it quantifies how much of the solution space that was unexplored gets covered by the generative model's queries. It is defined as follows, where we highlight that the ratio does not take into account the queries' frequencies, as a single occurrence has the same weight as a one that appears multiple times:

$$C = \frac{|g_{\text{sol}}|}{|\mathcal{S}| - T}. \tag{29}$$

One should expect the value of these metrics to depend on the number of queries Q that are retrieved from the trained model. For example, to

have a quality coverage of a space, i.e., $C \to 1$, one should have enough samples that fall in the entire unexplored space. However, this dependency does not constitute a limitation for drawing a comparison between models, as we can fix the number of queries for all the models under investigation, and evaluate and fairly compare their generalization performance at the given number of queries. Moreover, in 36, we further showcase the values of C as we increase the number of queries toward and beyond the size of the solution space. There is a clear trend towards the metric ideal limit $C \to 1$ as we increase the number of queries. Conversely, in Appnedix [] it is observed that fidelity and rate are not dependent on the number of generated samples, despite being sample-based metrics.

Note that the different metrics are not completely independent, as there are mutual relations between them. For instance, it can be noted that rate and fidelity are correlated, as $R = EF$. Rate is the same as fidelity whenever a model generates exclusively unseen queries, which only holds in the case of perfect generalization (or in pathological cases such as mode collapse to unseen and valid queries). Another example of mutual relation between the metrics is that $C \leq \frac{EQ}{|\mathcal{S}-T|}$, which implies that $C < E$ for large solution spaces and limited queries budget.

To further clarify the expected metrics' values for a well-generalizing model, we highlight that these metrics will be exactly 1 when evaluated for a model that exhibits the highest validity-based generalization. However, in a practical sense, this might be difficult to achieve; there exists a theoretical upper bound of 1 for all metrics, with the understanding that one should aim to reach this limit to obtain a robust model for generalization.

Model Behaviour	**E**	**(F, R, C)**	**Extra Check**
Perfect Generalization	1	(1, 1, 1)	N/A
Perfect Memorization	0	(null, 0, 0)	$\lvert d_{\text{gen}}\rvert \sim T$
Anomalous Pre-Generalization	~ 1	(0, 0, 0)	$\lvert d_{\text{gen}}\rvert \sim T$
MC (unseen and valid)	~ 1	$(1, 1, \sim 0)$	N/A
MC (unseen and invalid)	~ 1	(0, 0, 0)	$\lvert d_{\text{gen}}\rvert << T$
MC (seen and (in)valid)	0	(null, 0, 0)	$\lvert d_{\text{gen}}\rvert << T$

Tab. 1: **Metrics' values across various model behaviours.** The table displays the E and (F, R, C) values one obtains across different model behaviours such as perfect generalization, perfect memorization/overfitting, generating predominantly noise referred to as anomalous pre-generalization, and mode collapsing (MC) on various bitstring types. We see that F will be null in the cases where the number of unseen generated samples is zero. Additionally, we provide an extra check allowing to distinguish between cases in which the generalization metrics yield the same results.

Lastly, note that the pre-generalization condition in Equation 20

impacts the validity metrics; hence, exploration E is directly related to (F, R, C). For F, the pre-generalization condition in Equation 20 must be met in order for the metric to be well-defined. When the condition is not met, F will be null, and $C, R = 0$. Therefore, these metrics rely on the model's ability to go beyond the training set, and will indicate if the model is only data-copying. Other properties from the model can be inferred from these metrics as demonstrated in Table 1. For example, a metric which measures the degree of data-copying could be defined as $D = 1 - E$, hence perfect memorization would mean $E = 0$. In this framework, one can additionally use these proposed metrics to detect alternative and complementary behaviours to generalization and define additional metrics that are tailored towards specific properties one would like to investigate.

Evaluating Quality-Based Generalization To quantify the quality-based generalization properties of a generative model, we propose adequate metrics addressing the *sample quality* of the generated samples, which speaks to how many of the queries are more valuable results in the context of a specific application domain, i.e., how many bitstrings have a low enough associated cost. Since the quality of a result depends on a given cost function, this metric is task-specific, as opposed to the validity-based generalization case that only requires the notion of validity of a query, according to a well-defined hard constraint.

More precisely, there exists different nuances of this *sample quality* metric for the quality-based generalization assessment, providing two different versions with slightly different implementations of f in the right-hand side condition of Equation 22.

Firstly, we can consider the Minimum Value (MV) of the costs associated to the queries generated by the model as a relevant evaluation metric, since in many optimization applications the main goal is to find the solution that minimizes the cost, or equivalently, the sample with the best quality. This corresponds to choosing $f = \min$, so that the condition of Eq. 22 becomes:

$$x_g \text{ satisfies Eq. 21} \wedge \min_{x_g \in \mathcal{D}_{\text{Gen}}} c(x_g) < \min_{x_t \in \mathcal{D}_{\text{Train}}} c(x_t). \tag{30}$$

Despite its practical impact, this punctual metric can be highly unstable if it is not supported by enough statistics as the metric relies on generating one specific value, the lowest. Since generating the query with the lowest cost is highly dependent on the selected batch b of queries, we define this metric as an average across B batches of queries to avoid biasing the results

due to an anomalous batch. In other words, for each generative model evaluated, we define:

$$MV = \frac{1}{B}\sum_{b=1}^{B} \min_{x_g \in \mathcal{G}_{\text{sol}}^b} c(x_g).$$

Including such average in the definition of the MV metric itself contributes to alleviate its intrinsic instability, thus making it more robust for quality-based generalization evaluation.

Secondly, the Utility U can be defined as the average cost of a user-defined set P_t of unseen and valid samples from the generative model. Specifically, $P_t(\mathcal{D})$ is the set obtained from taking the $t\%$ of samples with the best quality (lowest costs) in $\mathcal{D}$. Setting $t = 5$, this corresponds to choosing $f = \cdot$ on the set P_5, and the condition of Equation 22 reads:

$$\begin{gathered} x_g \text{ satisfies Eq. 21} \wedge \\ c(x_g)_{x_g \in P_5(\mathcal{G}_{\text{sol}})} < c(x_t)_{x_t \in P_5(\mathcal{D}_{\text{Train}})}. \end{gathered} \tag{31}$$

Given its set-based definition, this metric is much more stable than the previous one.

Lastly, note that it is possible to give another definition of *sample quality*, which simply consists in counting the number of unseen and valid queries whose cost is lower than a specific critical cost value $c'(x)$ in the training set. For example, one could take $c'(x)$ to be the lowest cost value in the training set i.e., $c'(x_t) = \min_{x_t \in \mathcal{D}_{\text{Train}}} c(x_t)$. When utilizing this estimator, one is interested in verifying the following condition:

$$\Big|\{x_g \text{ s.t. } c(x_g) < c'(x_t)\}\Big| > 0, \text{ for } x_t \in \mathcal{D}_{\text{Train}}, \tag{32}$$

where clearly a higher value of the left-hand side implies a better sample quality. Even though this quantity can carry interesting information, we don't include it among our quality-based generalization metrics as it is a harsh restriction to impose and may only be important for optimization tasks that are looking for many potential MV bitstrings. This framework is not limited to the metrics proposed so far, but allows one to define several other figures of merit which can be relevant for specific applications at hand.

These metrics introduce insights into a model's quality-based generalization capabilities, and determine which models are able to generate the most value for task-specific challenges. Again, this approach can be utilized

beyond cost minimization problems, as long as there is a quantitative quality scale associated to each bitstring in the valid subspace.

5.3 TNBM vs. GAN

5.3.1 Learning Task

To demonstrate a practical application of the approach, we run the TNBM and the GAN, discussed in Section 3.2, on an important use case in the finance sector that addresses the challenge of cardinality-constrained portfolio optimization. The goal of such task is to minimize the risk σ associated to a collection of assets, randomly selected from the S&P500 market index, for a fixed desired return ρ. Below, we highlight how this task is amenable to the framework and requirements described in Section 5.2.2.

Given a fixed size N of the asset universe, a portfolio candidate can be encoded into a bitstring of length N, where each bit corresponds to an asset either being selected in the portfolio (1) or left out of the portfolio (0). Therefore, the search space $\mathcal{U}$ of all possible portfolios grows exponentially with the asset universe size, i.e. $|\mathcal{U}| = 2^N$.

To assess validity-based generalization within this task, the solution space $\mathcal{S}$ is comprised of all bitstrings containing a fixed number $k = N/2$ of selected assets, i.e., a candidate solution must be a bitstring with a fixed Hamming weight equal to k. With such k-cardinality constraint, the problem solution set $\mathcal{S}$ contains all possible portfolio bitstrings x that fit this constraint. Thus, its cardinality is:

$$|\mathcal{S}| = \binom{N}{k}. \tag{33}$$

To further assess quality-based generalization, an objective function is defined that encodes the quality of each bitstring, namely the financial risk σ associated to each portfolio, which in the case of the Mean-Variance Markowitz model [94] can be efficiently computed by means of Mixed Integer Quadratic Programming (MIQP) [70]. Unlike when investigating validity-based generalization, we use σ to re-weight the training dataset with the softmax function described in Equation 34.

As such, this task satisfies both the previously introduced conditions necessary to evaluate validity-based and quality-based generalization. This framework can be applied to any task that meets the essential requirements in Section 5.2.2, and is not limited to this financial application.

5.3.2 Simulation Details

For our experiments, we consider a specific instance of a cardinality constrained portfolio optimization task, where we aim at minimizing the associated risk σ for a given target return $\rho = 0.002$, such that the asset universe from which one can pick to build a new candidate portfolio has size $N = 20$. Here, assets are randomly selected from the S&P500 index, as previously done in Refs. [49, 70], and the return level ρ is the same as used in previous studies. We impose the cardinality constraint that each portfolio must have a fixed Hamming weight $k = N/2 = 10$. As previously stated, such an essential restriction creates a subset of the search space $\mathcal{U}$, of size $2^N \sim O(10^6)$, defining a solution space $\mathcal{S}$ of size $\binom{N}{k} \sim O(10^5)$. The choice of these values allows for a big enough space so that generalization capabilities can be probed.

Given the solution space of portfolio candidates, the data distribution $P_{\text{Target}}(x)$ given in Equation 23 used to assess validity-based generalization is automatically defined. To build a non-exhaustive $P_{\text{Train}}(x)$ as in Equation 24, only a fixed number $T = \epsilon|\mathcal{S}|$ of training samples are randomly selected from the solution space, thus making the task of learning the distribution $P_{\text{Target}}(x)$ highly non-trivial (despite it being defined as a uniform distribution over the valid bitstrings). Specifically, all generative models are trained for a fixed number of epochs $n_{\text{epochs}} = 100$ with a fixed value of T that equals 1% of the solution space (i.e., $\epsilon = 0.01$) , leaving the remaining 99% of the space available for testing generalization capabilities. Several values of this hyperparameter have been investigated, and we found this particular percentage to be a good choice as it gives the models many chances of generalizing, while providing enough samples $T \sim O(10^3)$ for the learning process to be successful. In order to assess quality-based generalization, we conduct the same process outlined above, with the addition of a pre-processing step that uses a softmax function to introduce risk-based information in the training dataset, so that low-risk portfolios are assigned a higher probability, and sampled with higher frequency.

We investigate the generalization behaviours of different versions of the TNBM and GAN architectures, using various hyperparameter sets. In the case of the TNBM, we consider different values for the bond dimension α, as this is the main parameter that affects the model quality. For GANs, the choice of hyperparameters is significantly more challenging [95]. Therefore, in addition to identifying hyperparameters via a trial-and-error procedure,

we investigate whether automated hyperparameter optimization using Optuna [96] could significantly improve the performance. We propose three different GANs that only differ in their hyperparameters as shown in Table 2, and show generalization behaviours for all of them. From here onward, we will refer to a GAN that has mode collapsed onto one seen and valid bitstring as GAN-MC and to the Optuna enhanced GAN as GAN+.

Hyperparameter	**GAN**	**GAN-MC**	**GAN+**
Prior Size	20	8	12
Hidden Size (G)	20	6	6
Number of Layers (G)	1	4	1
Learning Rate (G)	0.02	0.051	0.001
Hidden Size (D)	20	9	9
Number of Layers (D)	1	3	1
Learning Rate (D)	0.02	0.008	0.006
Negative Slope (D)	0.02	0.007	0.010
Dropout (D)	10^{-5}	0.024	0.107
Batch Size	50	71	56

Tab. 2: **GAN hyperparameter values.** The values labelled with $G(D)$ refer to the generator(discriminator). The hidden size indicates the number of nodes in each hidden layer within G and D, approximated to the same significant digit.

As mentioned above, all models have been trained for a fixed number of epochs and the associated generalization metrics have been computed based on a fixed number $Q = 10^5$ of queries retrieved from the trained model returned after the last epoch. Other strategies can be employed, such as considering the set of weights associated to the lowest loss function during training, or including more advanced training techniques such as early stopping. We decided to leverage a simple training scheme to avoid introducing any training bias and allow for the fairest comparison of the two models under examination. We also chose to sample this high magnitude of queries since this was not a limitation for the problem size considered here. However, in Appendix 10.1.3 we present the behaviour of our sample-based metrics as a function of the number of queries. All of the numerical experiments in this work were carried out with Orquestra® [8] for workflow and data management.

[8] https://www.orquestra.io/

5.3.3 Metric Robustness

The first step to validate our approach consists in showing the robustness of our sample-based metrics. To verify this, we conduct a statistical analysis of the generalization metrics' values and investigate the statistical errors associated to them. In addition, we propose an initial numerical investigation of the relationship between the values of our sample-based metrics and the distance measure from the model's distribution to the ground truth data distribution, in order to understand how the models' performance benchmarking approach connects with that of computational learning theory.

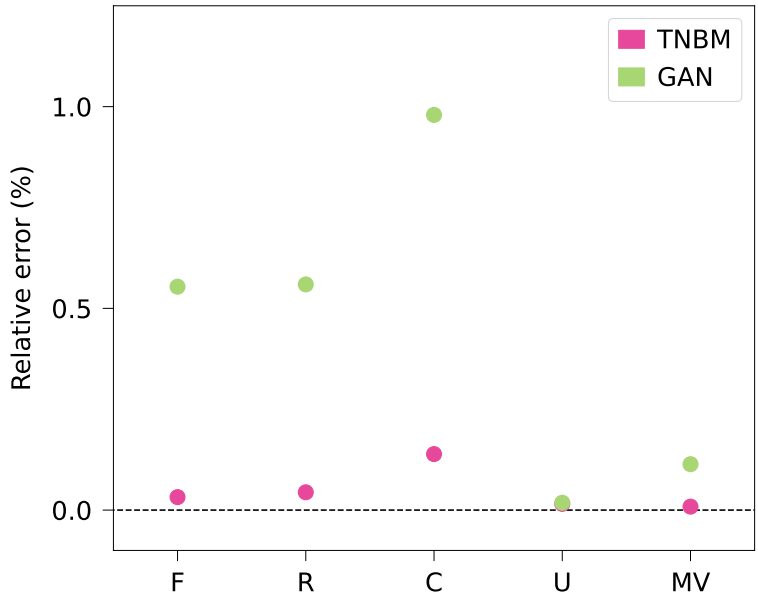


Fig. 8: **Robustness of the generalization metrics.** The plot shows the relative percentage error associated to each of the generalization metrics proposed in Section 5.2.1, listed on the x axis. The errors are estimated as the relative standard deviation of independent metric values computed on 30 sets of queries generated by trained TNBM (pink) and GAN (green) models. The proposed metrics show their statistical robustness: the associated error is small, suggesting that our approach is sample-based but not sample-dependent. Henceforth, new independent same-size query batches from the trained model will produce similar metric results.

We focus the robustness analysis on one instance of each of the two generative models presented in Section 5.3.2. Specifically, we consider a TNBM model with fixed bond dimension $\alpha = 7$, which has proven to be a good choice for generalization purposes as will be explained in Section 5.3.4. For GAN, we consider the set of hyperparameters displayed in the first column of Table 2, which were selected as reasonable values via a trial and error procedure (i.e., without leveraging automated hyperparameter optimization). The analysis can be extended to other instances to further strengthen the evidence of the robustness of our metrics.

After training these two model instances using gradient-based optimizers (see Table 2), we perform 30 independent query retrievals and compute our generalization metrics on these distinct sample sets. We then evaluate the relative percentage error[9] associated to each of the metrics to assess their statistical robustness. For each of the two models, the error values for both validity-based and quality-based metrics are shown in Figure 8.

The errors associated to the different metrics assume similar values for the TNBM and GAN: this supports our claim that our metrics are model-agnostic and can be used to evaluate generalization capabilities for any generative model of interest. Furthermore, we can see in Figure 8 that the relative errors are less than 1%, thus suggesting that our metrics show significant robustness when computed on different sets of queries. Hence, we can affirm that the metrics proposed in this work are sample-based but not sample-dependent across different query batches of the same size.

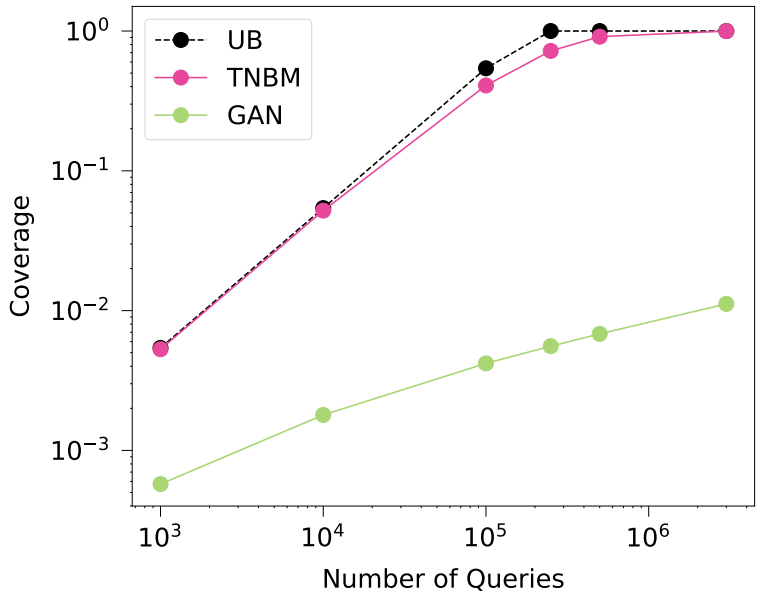


Fig. 9: **Coverage trends for increasing number of queries.** The plot displays the behaviours of the coverage metric for both TNBM (pink) and GAN (green) as we increase the number of queries Q retrieved from the trained models. The dashed black line shows the upper bound UB for each number of queries selected - i.e., the number of queries selected over the total size of the solution space. In the case of the TNBM, we observe that the coverage value follows the UB curve and saturates to the ideal value of 1 for large numbers of Q, corresponding to the scenario in which the trained model is able to generate all unseen and valid samples. In the case of the GAN, we still observe that the coverage value gets closer to UB and the ideal threshold of 1 when more and more queries are drawn from the model. However, it remains further from UB and never reaches the desired threshold, suggesting that our GAN requires more queries than the TNBM to be able to reach all the unseen samples in the solution space.

The latter statement requires further clarification in the case of the coverage metric in Equation 29. In this case, even though the coverage

[9] Relative percentage error is defined as the standard deviation of the metric values over their average.

does not depend on the set of queries, it does depend on the number of queries that are retrieved from the trained model. The ideal coverage value of 1 is reached in the limit of a large number of queries, when the trained model has the opportunity to generate enough samples to cover most of the solution space. However, we note that, given a query budget Q, the effective upper bound UB to the coverage value is set by

$$UB = \frac{\min(Q, |\mathcal{S}|)}{|\mathcal{S}|} \leq 1,$$

thus implying that the ideal value of 1 can be reached only with a sufficiently high number of queries, i.e., $Q \geq |\mathcal{S}|$. We investigated if the models considered so far show this trend as we increase the number of queries retrieved after training from 10^4 to $3 \cdot 10^6$. The results of the simulations are displayed in Figure 9; in Appendix 10.1.2] we compare them with the baseline given by random sampling from the search space $\mathcal{U}$. Results for how the other metrics vary with the number of queries Q are shown in the Appendix 10.1.3.

The data shows that the TNBM coverage closely resembles the UB trend for any given value of Q and saturates to the ideal value of 1 for a large enough number of queries, implying that this model is able to achieve excellent coverage. Conversely, the GAN coverage is further from the UB and slowly increases without getting to the desired threshold, thus suggesting that significantly more queries would need to be taken to achieve a perfect coverage of all the unseen and valid patterns. Since there is no guarantee that the desired threshold is reached with a finite number of queries, this result might as well indicate that the model is quite poor at generalizing due to a high number of unreachable patterns. This is particularly relevant in the case of very large solution spaces $\mathcal{S}$. In this circumstance, the coverage metric has an intrinsic limitation: its low value might indicate that the number of generated queries is insufficient ($Q << |\mathcal{S}| - T$), rather than being due to poor generalization ($|g_{\text{sol}}| \approx 0$). Therefore, in order to mitigate the above issue when evaluating single models in the case of large problem sizes, we envision the denominator in C to be replaced by the number of queries Q. This solution will slightly distort the meaning of coverage in Equation 29 to a new metric quantifying the rate at which the model generates unique unseen and valid samples. When extending to large problem sizes, we see this as a more relevant evaluation metric as one cares more about the diversity of unique unseen and valid samples the model can reach rather than reaching all of them, which would be impossible without the number of queries being at least

the size of the solution space. However, as our experiments are conducted with a mid-sized problem space, we stick to the definition in Equation 29 for our evaluation.

Even though the coverage metric is dependent on the number of queries and its interpretation in terms of generalization is affected by the size of the solution space, we can draw a fair comparison between the coverage of different models. Indeed, we can compare TNBM and GAN models if we keep the number of queries generated from each fixed, as reported in Section 5.3.5, where it will be shown that the quantum-inspired model outperforms this GAN model when given the same sample budget.

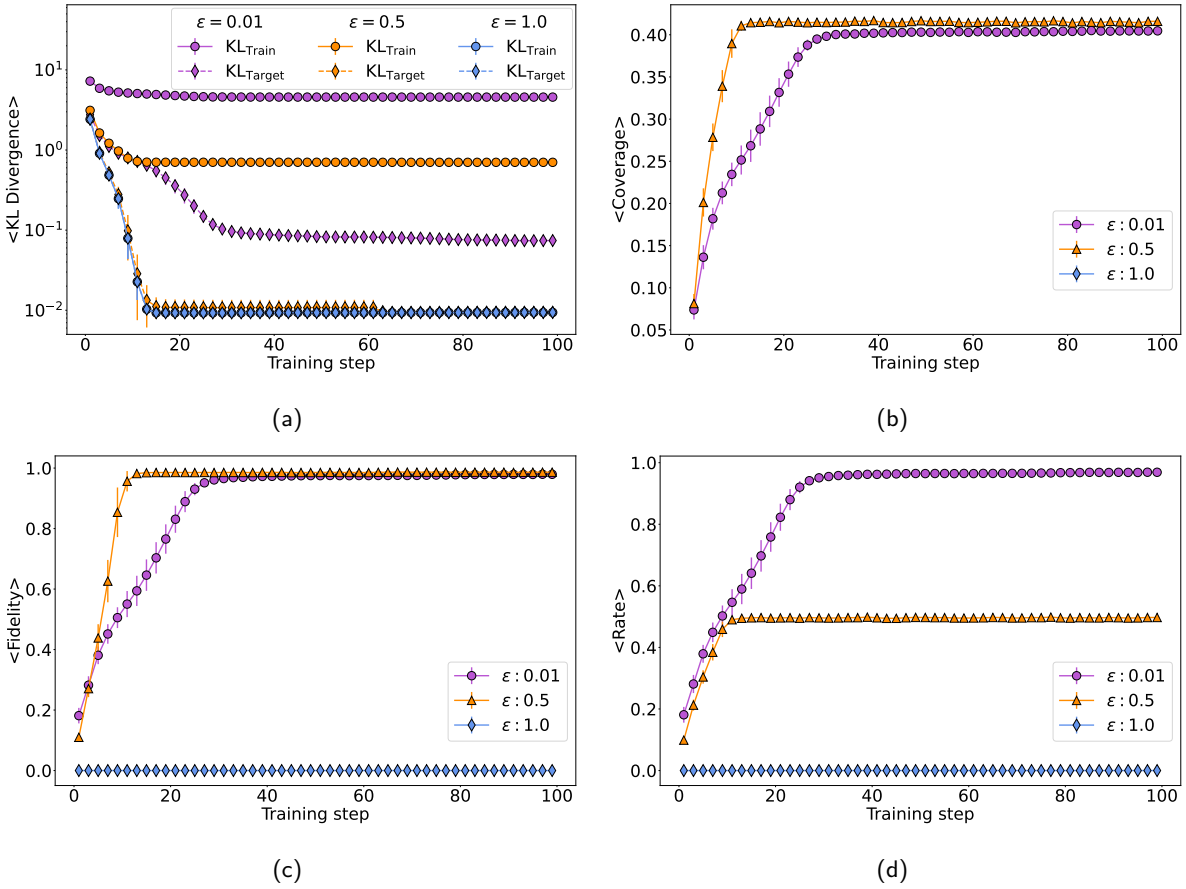


Fig. 10: **TNBM's generalization performance throughout training across various ϵ values.** Here, we show the relationship between the model's ability to learn the ground truth distribution with access to a restricted portion ϵ of the solution space, and the (F, R, C) values computed throughout the model's training. In panel (a), we see the KL_{Train} is always higher than the KL_{Target} across ϵ values - indicating good inference performance. Panels (b), (c), and (d) show that the model's (F, R, C) values increase throughout training (note that coverage is not defined (nan) for $\epsilon = 1$). The concurrent relationship between approximating the ground truth and obtaining high (F, R, C) values suggests a positive correlation between our practical models' performance benchmarking approach and computational learning theory.

Lastly, we put forth an initial investigation on the correlation between our metrics and the model's ability to infer the ground truth, as is the goal in computational learning theory discussed in Section 3.3.1. In Table 3,

we report the average values of (F, R, C) that result from five independent trainings of TNBMs with $\alpha = 7$. To take into account the fact that we span over a few ϵ values, we also show a normalized version of the rate value, given by $\tilde{R} = R/(1-\epsilon)$. Alongside the (F, R, C) values, we record two versions of the KL divergence: the quantity KL_{Train}, computed as usual between the model's output distribution and the training distribution in Equation 24, and the quantity KL_{Target}, computed between the model's output distribution and the uniform ground truth data distribution in Equation 23. Note that the latter is not usually available in real-world scenarios, since the ground truth is unknown; however, we find it relevant to analyze this quantity to validate our practical approach to generalization by relating it to computational learning theory. We see that with access to very little data ($\epsilon = 0.01$), the model yields high (F, R, C) values and gets closer to the data distribution than the training distribution - as $KL_{\text{Target}} < KL_{\text{Train}}$. When we increase ϵ to half of the solution space, we see that the (F, R, C) metrics increase and the model is also able to approximate the ground truth more closely, since KL_{Target} decreases. Hence, we see a promising correlation between our metrics' values and the model's ability to infer the ground truth in both of these data regimes.

Metric	$\epsilon = \mathbf{0.01}$	$\epsilon = \mathbf{0.5}$	$\epsilon = \mathbf{1.0}$
F	0.979(0.38%)	0.986(0.06%)	0.0
R	0.969(0.39%)	0.497(0.28%)	0.0
$\tilde{R}$	0.979(0.39%)	0.993(0.28%)	nan
C	0.405(0.52%)	0.416(0.32%)	nan
KL_{Train}	4.575(0.07%)	0.702(0.01%)	0.009(0.36%)
KL_{Target}	0.074(13.28%)	0.009(0.56%)	0.009(0.36%)

Tab. 3: **The relationship between the validity-based metrics and learning the ground truth for the TNBM**. Down each column, we record the final average (F, R, C) metrics' values (including the normalized rate $\tilde{R}$) along with the average KL divergences of the model output distribution relative to the training distribution, denoted as KL_{Train}, and to the data distribution, denoted as KL_{Target}. We see that there is a good correlation between the high-scoring metrics' values and learning the ground truth distribution, even in multiple data regimes. We see that the largest discrepancy between the two frameworks exists when $\epsilon = 1$, where $KL_{\text{Target}} = KL_{\text{Train}}$ reaches a low value, but the other metrics are either zero or undefined. This is a case of *memorization*, where the model still scores high in the context of learning the ground truth, while demonstrating poor performance from a practical generalization standpoint. This is expected from a practical perspective: the generative model cannot add value in terms of generating novel samples, since all of them were given as part of the training set. All relative percentages errors are computed across five independent trainings.

The main discrepancy between the two approaches occurs when the model is provided all of the data during training ($\epsilon = 1$). In this case

that, we see that $KL_{\text{Target}} = KL_{\text{Train}}$, and thus there is no room for generalization to occur, as defined in Section 5.2.1. Therefore, the metrics' values are either zero or undefined (nan) in this instance. Despite this, we still see that the model is able to learn the ground truth well, as indicated by a low KL value. The ability to assess this *memorization* behaviour is the main distinction between our practical approach in evaluating generalization and the one utilized in computational learning theory. From a practical standpoint, being able to identify this behaviour is highly relevant, thus supporting the need for a more practical approach to generalization to be considered in parallel to the theoretical one. In Figure 10, we show multiple plots that report our metrics' values throughout the entire training alongside the $KL_{\text{Train}}, KL_{\text{Target}}$ values for a more complete analysis. Remarkably, the different panels in Figure 10 demonstrate excellent correlations between the theoretical and practical approaches, while also highlighting the value of having a multidimensional evaluation perspective, which provides enhanced explainability when assessing strengths and weaknesses of generative models. We note that while this example indicates a good correlation between our metrics' values and the ground truth inference ability, more investigations are necessary to strengthen the understanding of this relationship, potentially including theoretical proofs that establish precise connections between the two approaches.

5.3.4 Spotting Pitfalls in Generative Model Training

We further demonstrate that we can use our metrics to detect common pitfalls that are known to affect the training of the TNBM and GAN models. This result strengthens the validity of our approach, which turns out not only to be a good framework for quantifying generalization of generative models, but also to enhance the study of their trainability. In the following sections, we show an example of this study for each of the models. For the TNBM, we analyze the relation between the bond dimension α, our generalization metrics, and the trainability of the model. Conversely, for the GAN, we investigate the relation between our metrics and mode collapse. Additional results to compare the training stability of the two classes of models are shown in Figure 40 in Appendix 10.1.1.

TNBM Bond Dimension and Trainability In the TNBM architecture, the bond dimension α of the MPS plays an important role in the model's ability to generate good quality samples as it is directly correlated with the

expressive power of the model. Typically, increasing the bond dimension leads to a better model approximation. We take this one step further and directly connect bond dimension to the model's generalization behaviour and trainability.

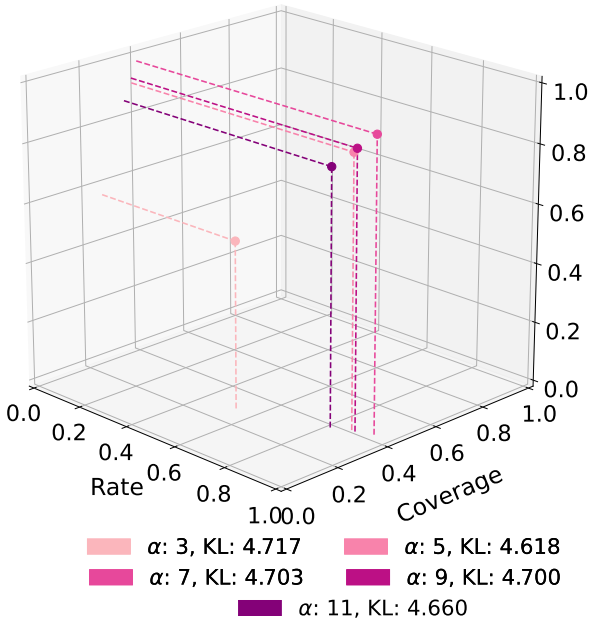


Fig. 11: **Training and generalization behaviours of the TNBM with different bond dimensions.** The plot displays the 3D evaluation of the validity-based generalization capabilities of the TNBM models with various $\alpha \in \{3, 5, 7, 9, 11\}$. Each data point corresponds to the average metrics' values, whose associated error is too small to be visible on the plot. The legend connects each α to the last KL divergence value in the training after 100 epochs. The plot demonstrates that for various α values, there is a connection between KL divergence values of the model distribution to the training distribution, thus establishing a link between this capability and trainability properties of generative models.

In light of this goal, we train five different instances of the TNBM architecture on a fixed training dataset with various bond dimensions $\alpha \in \{3, 5, 7, 9, 11\}$. For a given α value, we select a typical[10] training and build a model with the last set of parameters retrieved after the learning process. We then generate 15 independent query batches from the trained model and compute our validity-based generalization metrics (F, R, C). We show the results in Figure 11, where we display the average metric evaluations for each bond dimension α. In the plot legend, we report the last loss function value during training (complete training loss curves can be found in Appendix 10.1.1.

From Figure 11, it can be seen that the median value of the KL divergence occurs for $\alpha = 7$: this result motivates the usage of such value

[10] A typical training instance is identified as the resulting model from the median value of the loss function (i.e., KL divergence) at the last epoch, out of 30 independent trainings.

in Section 5.3.3, as it suggests that the training is most typical for this choice of the hyperparameter value. It is not surprising that the lowest value of the loss function, obtained for $\alpha = 5$, does not correspond to the best validity-based generalization performance, as shown in Figure 11, because this loss is relative to the training rather than the data distribution. If the model was to perfectly fit the training distribution, we would see data-copying rather than generalization behavior - which is a form of overfitting. We expect that our metrics will be able to identify similar overfitting behaviours when associated to an extremely successful training curve (Table 1).

As the bond dimension grows, we see an increase in (F, R, C) up to $\alpha = 7$, and then the metrics' values begin to decrease. Thus, it seems that we are hitting a trainability *Goldilocks region* around $\alpha \approx 7$, with $\alpha < 7$ leading to underperforming models and $\alpha > 7$ being too expressive for the model to be able to generalize successfully. These results demonstrate that we can use our metrics to identify thresholds in hyperparameter tuning and to get insights on the trainability of the model as it relates to generalization.

Mode collapse in GAN One of the major issues that affects GAN training is the so-called mode collapse behaviour [97]. This undesired phenomenon occurs when the generator learns to produce a very limited number (sometimes only one) of highly plausible outputs, thus affecting the ability of the generative model to further explore the solution space. Since mode collapse is a well-known pitfall, several strategies have been proposed to mitigate this issue in the context of GANs, among which a promising algorithm is the Wasserstein GAN [98, 99].

We propose an example of how the metrics are able to detect mode collapse, when it occurs. We fine tune our hyperparameters such that the GAN exhibits mode collapse behaviour (see details in Table 2 in Appendix 10.1.1) for a fixed training dataset. We run a typical[11] training of this GAN-MC architecture, and then sample 15 query batches from the trained model to compute our generalization metrics (F, R, C).

We display the validity-based metrics for the GAN and GAN-MC in Table 4. For the GAN-MC, we see that fidelity and rate are the ideal value of 1, thus suggesting that the model generates exclusively unseen samples with the desired cardinality. However, the coverage value is close to 0, thus

[11] A typical training instance is identified among 30 independent trainings as the one whose mode collapse shows the correct cardinality and whose last associated Hausdorff distance [**?**] during training is the median. We highlight that the training is performed via an adversarial strategy, hence we use the Hausdorff distance only as a figure of merit to monitor the training.

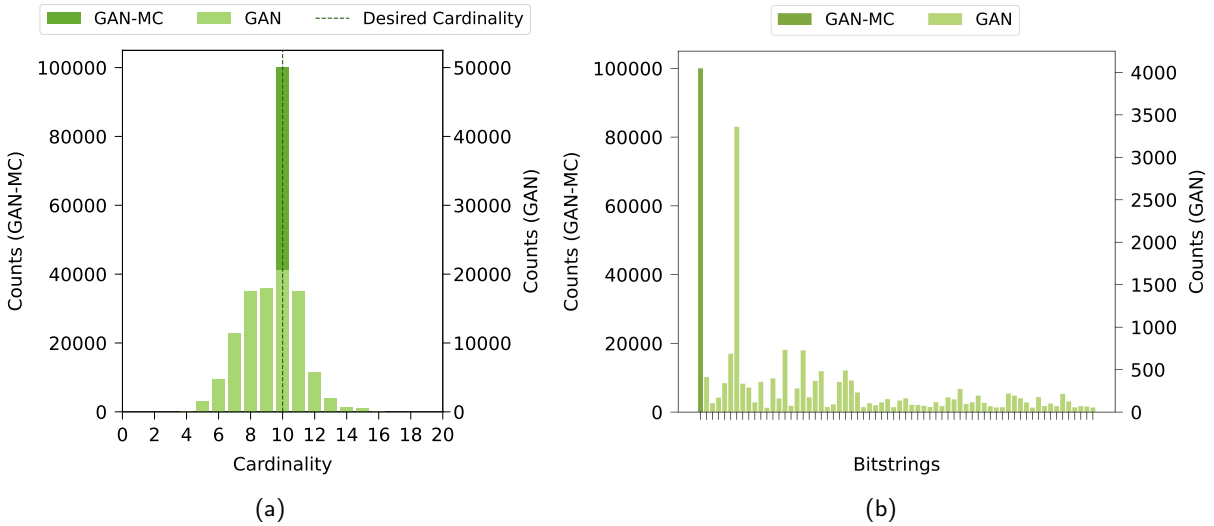


Fig. 12: **Visualization of mode collapse in GAN training.** Figure 12a shows the cardinality distribution of generated queries for GAN and GAN-MC, indicating that GAN-MC produces only samples with the desired cardinality (dashed line), whereas the GAN queries populate a larger subset of the cardinality domain. Hence, GAN-MC is associated to perfect fidelity $F = 1$ and rate $R = 1$. However, in Figure 12b the queries' diversity is displayed, where the x axis represents the set of distinct generated bitstrings (for readability, bitstrings labels are not shown, and only bitstrings with counts > 50 have been included in the histogram). We can see that GAN-MC always generates the same unseen and valid query (mode collapse phenomenon), as opposed to GAN, which is able to cover a significantly larger portion of the solution space, as reflected by the metrics' value in Table 4. Note the different scales for the y axes in both Figure 12a and Figure 12b.

it is far from its ideal threshold, since the model is only able to produce one single pattern and does not have the ability to explore the solution space and cover it as much as possible. Such anomaly in the validity-based generalization metrics' values is not present if the training of a GAN doesn't exhibit training pitfalls, as displayed by the GAN results in the same table.

We note that these metrics' values only capture mode collapse behaviour for models that collapse onto an unseen and valid bitstring. If the model were to collapse onto a seen bitstring (in-training mode collapse), F would be not well-defined and both C and R would equal zero. These metrics' values would be indistinguishable from the perfect memorization regime. In order to avoid this, one should also compare the number of individual queries generated, $|d_{\text{gen}}|$, to the size of the training set T. This would provide the additional information necessary to detect any form of mode collapse. Expected metrics' values for various mode collapse behaviours along with other model training pitfalls are displayed in Table 1. In summary, our metrics reflect mode collapse upon occurrence and therefore they can provide insights on the training progress of generative models.

In order to better visualize the difference between the two aforementioned

models and detect the mode collapse phenomenon, in Figure 12a we display the cardinality distribution of the generated queries for the two GAN variants under examination: for GAN, the distribution is centered around the correct cardinality but shows a larger spread as compared to the case of GAN-MC, where all the queries satisfy the cardinality constraint. Nevertheless, Figure 12b allows one to identify the occurrence of mode collapse onto an unseen and valid bitstring: the GAN-MC model generates always the same query, as opposed to the diversity of samples retrieved from GAN.

These results demonstrate that we can use our metrics to identify the occurrence of a very well-known pitfall affecting the learning process of GANs, thus providing an insightful tool for the challenging task of monitoring the training of generative models.

5.3.5 Evaluating and Comparing Models

We use our quantitative metric-based approach to evaluate the validity-based and quality-based generalization capabilities across different generative models and compare their performance.

We run 30 independent trainings for a fixed training dataset and choose the best run, which we define as the run with the lowest loss function at the end of the trainings. Then, we generate 15 query batches from such trained model, for each of the generative models under examination. We note that while we use a fixed training dataset to compare models, this evaluation method holds across multiple training datasets that could be selected from a specific problem instance. Indeed, each dataset is characterized by the same asset universe, cardinality, and seen portion ϵ, but different datasets can be built by simply uniformly drawing independent bitstring subsets from the support of $P_{\text{Target}}(x)$. We perform this analysis in Appendix 10.1.3, showing that validity-based and quality-based generalization metrics for 15 different training datasets display similar values, thus showcasing the robustness of the models' behaviour, and the conclusions shown in this work.

For validity-based generalization, we construct $\mathcal{D}_{\text{Train}}$ by sampling from a $P_{\text{Target}}(x)$ that is uniform over the solution space of cardinality-constrained bitstrings, whereas for quality-based generalization, $\mathcal{D}_{\text{Train}}$ is re-weighted with cost-related information, i.e., from $P^{(\text{w})}_{\text{Train}}(x)$, as in Equation 34. As stated previously, we use one fixed dataset for our evaluation in Section 5.3.2. Post training, $Q = 10^5$ queries are collected from each model for comparison.

Validity-Based Generalization We first show the validity-based generalization results for each type of model. While we present these results as both an evaluation and comparison of models, we would like to emphasize that our results do not speak for all GAN or TNBM models, as each type of model may contain various hyper-parameters, multi-layered architectures, and other variances that would lead to different results. We choose to focus on using these models to demonstrate the robustness of our framework and metrics, such that when exploring various GAN, TNBM, or alternative model architectures, this approach can be replicated.

Results for (F, R, C) are listed in Table 4, along with the values of the exploration E; the corresponding results for the metrics' baseline given by random sampling from the search space are reported in Appendix 10.1.2. Additionally, we visualize the average validity-based metrics in Figure 14 through a 3D representation. Lastly, Figure 13 gives an intuition of how the two models perform and allows to visualize their different abilities in reconstructing the data distribution $P_{\text{Target}}(x)$, showing the remarkable performance of the TNBM as reflected in the metrics' values.

In evaluating our models, we see that the TNBM is a clear winner with average values $(0.989, 0.978, 0.409)$. The model achieves near-perfect rate and fidelity. As the maximum coverage one can achieve is the number of queries over the size of the solution space ($UB = 0.54$), the TNBM performs remarkably well. Indeed, the ratio of the average coverage to the upper bound UB for the TNBM is high, i.e. $C/UB = 76\%$. However, we note that the upper bound represents a scenario that would rarely happen in practice, thus representing a pessimistic reference value. A more realistic reference can be derived if one considers the ideal expected coverage $\overline{C}$ when sampling from the data distribution $P_{\text{Target}}(x)$. By means of simple statistical considerations (see e.g., [100, 101]), it can be shown that

$$\overline{C} = 1 - (1 - \frac{1}{|\mathcal{S}| - T})^{Q},$$

and this estimator indicates which coverage C one should expect when the generative model has perfectly learned the data distribution and generates samples accordingly. When comparing the average TNBM coverage to this more realistic reference value, we obtain a surprisingly high value of 97%, which shows that the model has learned an extremely good approximation of the data distribution $P_{\text{Target}}(x)$. In Table 4, we include $C/\overline{C}$ values for all models in order to highlight how well each model's average coverage compares to the ideal expected coverage. The limit of $C/\overline{C} \to 1.0$ holds for models with perfect generalization.

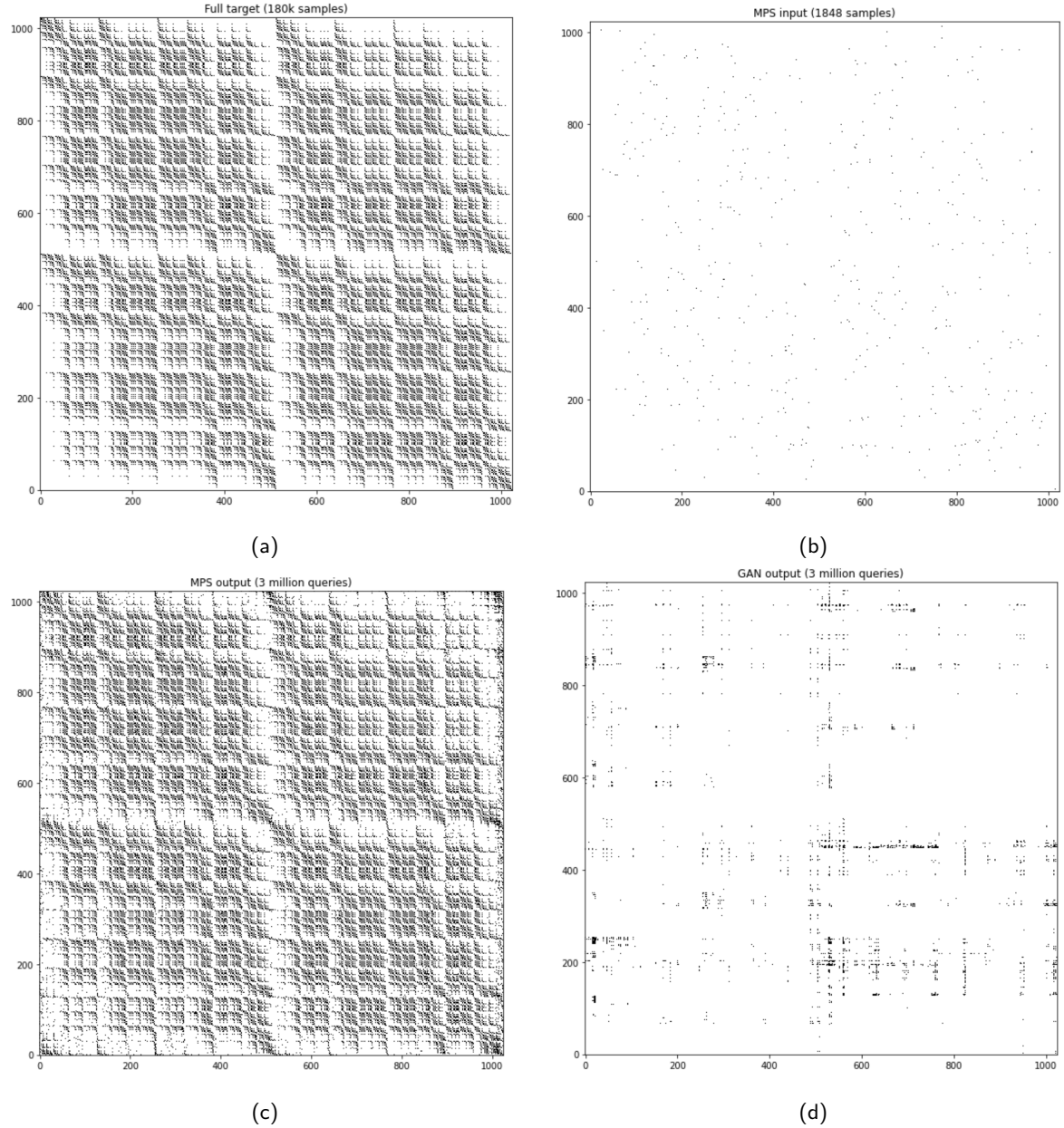


Fig. 13: **2D Visualization of distributions.** Figure 13a shows the 2D visualization of the exact data distribution defined by the solution space $\mathcal{S}$, where we see that a specific pattern emerges from the cardinality. In Figure 13b, we display the 2D visualization for the training distribution, where the same distribution was given to both the TNBM and the GAN models. As shown in Figure 13c, it is very remarkable that with this very limited number of training patterns provided to each model, the TNBM is able to generate the pattern from the data distribution almost exactly (as reflected in the metric values too). On the contrary, in Figure 13d we see that while the GAN is able to learn portions of the pattern, it struggles to reproduce this data distribution.

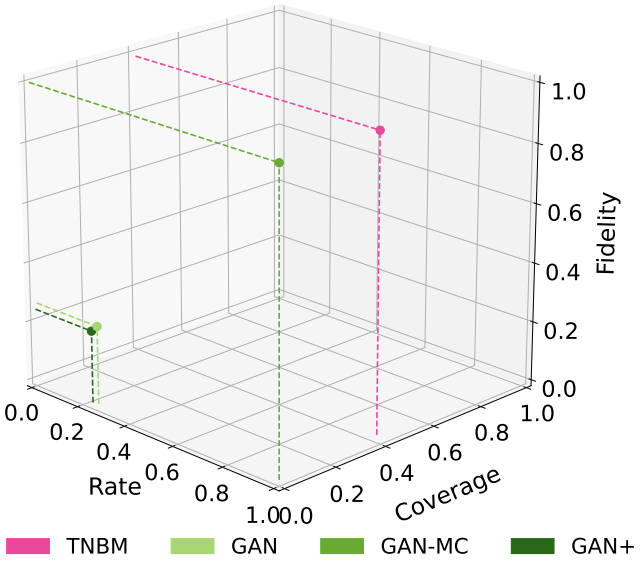


Fig. 14: **3D evaluation of validity-based generalization metrics for different generative models.** The plot displays results for four models, namely the TNBM with $\alpha = 7$ (pink), GAN (light green), GAN-MC (medium green) and GAN+ (dark green). The solid points show the average (F, R, C) values across 15 query batches, whose associated error is too small to be visible in the plot. We see that our TNBM is the clear winner compared to our GAN models.

As shown in Figure 9, the TNBM is able to achieve an improved coverage when sampling up to 3 million queries. The model has a high exploration rate of 98.9%, i.e. $E = 0.989$, such that most of the generated samples were not fed to the model during training. The GAN has much poorer average (F, R, C) values with a slightly higher exploration rate than the TNBM, thus showing that neither of them is performing mere data-copying. The GAN achieves metric values $(0.263, 0.261, 0.006)$, but 99.5% of its generated samples are outside of the training set. One can conclude that while the GAN has the potential to produce novel samples, it requires improved optimization strategies in order to avoid generating noisy samples - i.e. samples that do not match the cardinality constraint - so that fidelity and rate can grow to larger values. The GAN is not able to learn the underlying features as well as the TNBM, and thus is not able to generalize as well. Lastly, we compute the TNBM-to-GAN ratios for the validity-based metrics, and see that the TNBM is $(3.76, 3.75, 68.2)\times$ better than the GAN, respectively across (F, R, C) values. We would like to highlight that using metric ratios, rather than absolute values, allows one to have a clearer picture of the relation between different models, and this strategy is especially useful when considering the coverage, whose absolute value has been shown to be more heavily affected by the number of collected queries Q.

As explained in Section 5.3.4, we further show visually that our metrics detect mode collapse in GANs. The GAN-MC has an exploration rate of 100% ($E = 1$), demonstrating that the single generated sample was not introduced in the training set. Without the prior knowledge that the model exhibits mode collapse, we can use the average (F, R, C) values $(1.0, 1.0, 5.5\text{e-}6)$ to detect this behaviour. If perfect fidelity and rate are achieved, with a coverage near zero, we can conclude that the model has focused in too closely on one or a few unseen and valid bitstrings. In general, whenever $C \to 0$ we can safely identify the behaviour as mode collapse. Then, we consider the (F, R, C) values of the GAN+ and see that while the GAN+ is able to explore slightly more than the GAN, the (F, R, C) values are very similar, namely $(0.243, 0.243, 0.001)$, showing that the optimization scheme with Optuna doesn't bring a significant improvement for our specific GAN model in terms of generalization.

Metric	**TNBM**	**GAN**	**GAN-MC**	**GAN+**
E	0.989(0.02%)	0.995(0.02%)	1.0	1.0(0.003%)
F	0.989(0.03%)	0.263(0.6%)	1.0	0.243(0.4%)
R	0.978(0.03%)	0.261(0.6%)	1.0	0.243(0.4%)
C	0.409(0.15%)	0.006(1.7%)	$5.5 \cdot 10^{-6}$	0.001(2.5%)
$C/\overline{C}$	0.971	0.014	$1.0 \cdot 10^{-5}$	0.002

Tab. 4: **Pre-Generalization and validity-based generalization metrics for all models.** We display the average exploration E and the average (F, R, C) values for each best model run with an average and the associated relative percentage error across 15 query batches. All the models exhibit a high exploration rate, thus showing that data-copying is not occurring. We see that our TNBM model outperforms our GAN and GAN+ models by more than 70 percentage points for F and R. C is about 68x larger for TNBM than the GAN models. We further include the ratio of the coverage C to the ideal expected coverage $\overline{C}$ to highlight the large difference between the TNBM and the GAN's ability to successfully learn the underlying data distribution $P_{\text{Target}}(x)$. Additionally, for GAN-MC, we see perfect F and R and a near zero C value, indicating mode collapse behaviour. Note that no error is provided for the GAN-MC as all the models produce exactly the same values for the metric, except for the coverage whose associated error is negligible.

Lastly, we note that F and R are highly correlated for each trained model. This is the case only because in all of the models studied here the exploration E is quite high ($E \approx 1$). In this limit, and given that $R = EF$, then we have $R \approx F$. It is important to note that there is no reason to expect a value of E to be similar across all models, as it happened for the GAN and TNBM explored here.

Quality-Based Generalization We evaluate our generative models' ability to generate high quality samples using our quality-based approach and metrics. The models (TNBM and GAN) are evaluated across the two

sample quality metrics described in Section 5.2.3: Minimum Value (MV) and Utility (U). Note that for calculating the MV, as discussed in Section 5.2.3, five batches of $Q = 10^5$ queries were used. Hence, the total number of query retrievals used to compute this metric is $5\times$ the number of query sets one would desire for gathering statistics (in our case, $15 \times 5 = 75$ query sets, but this can be adjusted according to the available sampling budget).

When averaged over the 15 independent query retrievals, both the TNBM and the GAN meet the conditions in Equation 30 and in Equation 31, as shown in Table 5.

Metric	**TNBM**	**GAN**	**Threshold**
MV	0.1017(0.01%)	0.1024(0.17%)	0.1035
U	0.1049(0.017%)	0.1048(0.02%)	0.1059

Tab. 5: **Quality-based generalization metrics for TNBM and GAN models.** The first column shows values obtained by averaging over all 15 query retrievals for the TNBM's sample quality performance, along with the associated relative percentage error. The second column displays the metrics' values and relative percentage error for the GAN model. The last column displays the training threshold, defined as the MV and U computed for the samples in $\mathcal{D}_{\text{Train}}$. We see that both the TNBM and the GAN meet the conditions in Equation 30 and Equation 31.

We see that our TNBM exhibits a lower MV than our GAN, even though both beat the training set on average. Thus, our TNBM model shows slightly enhanced performance when searching for a minimum value of the cost function $c(x)$, which is assumed to be the financial risk $\sigma(x)$ in the specific application we are considering. While this may be relevant when one aims at finding the lowest possible minimum in an optimization task, it may not be the most important condition for alternative tasks that are simply looking for multiple low cost options - not necessarily the lowest. For example, when looking for a large frequency of low cost samples, the condition in Equation 31 may be more important and robust for comparing models. Note that the value of the utility threshold parameter t can be set according to the task at hand. In our task, we take $t = 5\%$ as an appropriate threshold for demonstrating the model's ability to obtain the tail-end of the distribution over low-risk samples.

From Table 5, we observe that the GAN and the TNBM have practically the same U, despite having such a large difference in (F, R, C) values. We conclude that while both models generate new portfolios that happen to be similarly low in risk when taking the smallest 5% of unseen and valid portfolio risks, the TNBM is simply able to generate more of them than the GAN (TNBM: 4556, GAN: 843, i.e., 5.4×). We display these

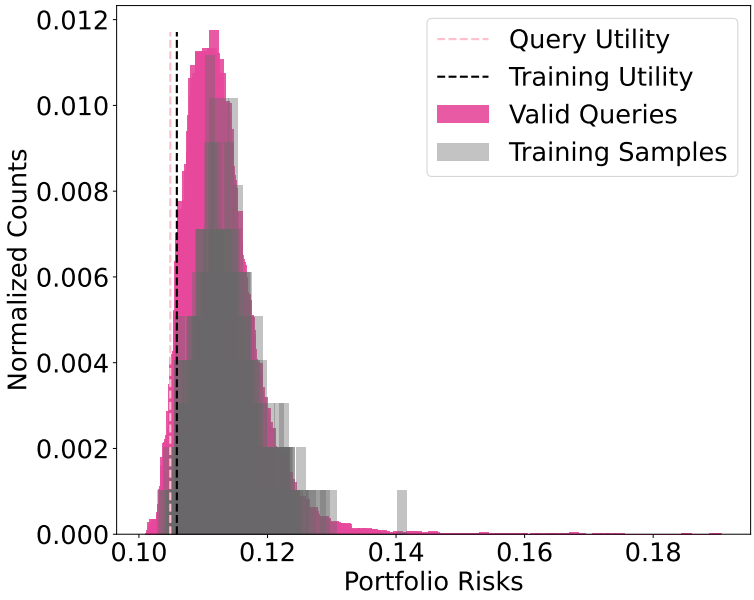


Fig. 15: **Visualization of quality-based metrics for TNBM-generated queries.** The plot displays the number of portfolio counts associated to given risk values. The pink spikes represent valid TNBM queries, whereas the gray spikes represent the samples from the training set. Note that for calculating our metrics, we used $Q = 10^5$ queries, but the training distribution only contains $O(10^3)$ samples. We normalize the counts on the y axis to provide a fair visual comparison between distributions, and we set the utility threshold to $t = 5\%$. Because the training distribution is re-weighted to favor lower risk values, the model distribution learns this feature in the dataset, and generates an even higher frequency of low risk values. The model queries have a lower utility (pink dashed) than the training set (black dashed), and the model is able to produce samples that have lower risks than those in the training set. We see that our TNBM model is able to effectively generalize to low-risk samples.

utility samples for TNBM in Figure 15, demonstrating the comparison of U relative to the training distribution P_{Train}. We include the same figure for the GAN in Appendix 10.1.4.

Hence, the GAN is able to generalize to similarly low risk portfolios as the TNBM, but fewer in number and less diverse than those of the TNBM. Our (F, R, C) metrics support that this generalization diversity is one of the largest differences between our TNBM and our GAN. Therefore, our TNBM model achieves superior performance when looking to produce a large diversified batch of new low-risk valid portfolios. We note that it remains an open question as to why the TNBM's performance is of such high quality. Investigating the nature of the model's inductive bias remains an ongoing research effort and opens an interesting opportunity to understand the power of quantum and quantum-inspired model when compared to their classical counterparts.

Lastly, we calculate the number of unique portfolios each model is able to produce that have a lower associated risk than a critical cost in the training set $c'(x)$. When this critical value is equivalent to the sample with

the lowest risk in the training set, our TNBM on average is able to beat our GAN with a 61:4 ratio. In other words, our TNBM model is able to generalize to 61 unique portfolios that have a lower risk than the lowest risk in the training set, while the GAN can only produce 4 (i.e., $\sim 15\times$). We introduce this condition in Equation 32 on top of the other two metrics in order to have an additional layer to determine whether a model is suitable for generalization. Note that one could adjust this critical cost threshold $c'(x)$ to relax the restriction. For example, when $c'(x)$ is equivalent to the risk taken at cutoff of the lowest 5% of samples in the training set, the TNBM-to-GAN ratio becomes 6709:345 on average (i.e., $\sim 19\times$).

While the model might meet the *sample quality* requirements Equation 30 and Equation 31, it might be poor at finding many samples with lower cost than $c'(x)$, which is not ideal when one is not only concerned with the global minimum, but also with generating a large quantity of low-cost samples. Our GAN works well under these requirements. On the other hand, our TNBM model shows good quality performance for generalizing to both valid and quality-based portfolios with high diversity and frequency.

5.4 Summary & Implications

In summary, the most prominent contribution of the research outlined in this chapter is the introduction and demonstration of a framework that unambiguously defines generalization-based practical quantum advantage in the generative modeling domain. Generalization is the gold standard for measuring the quality of a machine learning model. With generative modeling having an edge over supervised models in the race for quantum advantage [10], and we hope this work opens the possibility to start this race on a solid ground, and on datasets with commercial relevance [102]. This framework provides a strong definition of *what is good?* in the context of quantum generative modeling; it will be used in the Chapter 6 to evaluate the potential advantage of QCBM's (discussed in Section 4.2.2) and it has already been used in subsequent works [103] and [104].

As shown here, training GANs and other state-of-the-art classical generative models can be challenging to the point that we report a superior performance from the quantum-inspired generative models used here. Although we expect potentially better results from other classical proposals, there is room as well to improve the quantum-inspired versions explored here. We hope this work incites both quantum and classical ML experts to use this framework to enhance the performance and design

of their models, in this now quantitative race towards demonstrating practical quantum advantage in generative modeling.

6 ARE QCBMS GOOD GENERATIVE LEARNERS?

6.1 Background

One of the most popular quantum circuit families for generative tasks, known as Quantum Circuit Born Machines (QCBMs) [20], have demonstrated remarkable capabilities in modeling target distributions for both toy and real-world datasets [20, 26, 74, 105, 106, 70, 107, 108, 28, 75]. It has been shown that these models have the ability to express distributions that are difficult for classical probabilistic models [21, 73, 109, 30, 110], further motivating an investigation into these models for quantum advantage applications.

So far in the literature, assessing the quality of these QCBM's, discussed in Section 4.2.2, has been almost entirely limited to how well they can memorize or reproduce a known target distribution. Despite its intrinsic value for benchmarking purposes, by sticking to a reproducibility/data-copying metric such as minimizing the empirical Kullback-Leibler (KL) divergence [77] or negative log-likelihood (NLL - shown in Equation 11) as the primary method of evaluating a generative model, we are not properly assessing the model's true ability to generalize - and hence, its true potential for *quantum advantage*.

The clear question to consider is: *Do QCBMs provide an advantage?*; or circling back to the much larger question *Are these widely accepted models good generative learners?*

More recent works have attempted to evaluate QCBM's learnability from a data complexity perspective. For example, Ref [111] specifically claims to focus on a learning theory, deriving theoretical generalization bounds for the model trained via Maximum Mean Discrepancy; however, their approach (based on Ref. [112]) does not explicitly take into account the crucial feature of novelty, which constitutes an essential ingredient to define advantage, as discussed in Chapter 5. The generalization error proposed in [111] quantifies the deviation of the empirically optimized probability distribution encoded by a model after training from the best probability distribution the model can represent, given its expressivity and a desired target distribution [12]. When this deviation is minimized, the learning performance is maximized. However, if a generative model is simply memorizing data, such deviation will be exactly zero, hence implying that this metric is not sensitive to the novelty of the generated

[12] In practice, the target distribution is unknown, so including it in the model evaluation process is only feasible from a theoretical perspective.

samples. Since the comparison between training set and generated queries is out of the scope of Ref. [111], the model's true ability to generalize is not fully assessed. Also, recently, theoretical rigorous results were obtained for certain output distributions from local quantum circuits, proving some challenges in achieving a separation advantage with respect to classical models within this family [31].

In other words, most proposals fail to conduct a complete investigation as to how the QCBM learns features of complex target distributions from a finite set of training data, and as a result ignore the role that important resource requirements (e.g., circuit depth and amount of training data) play in understanding the model's generalization capabilities in tasks of practical relevance. In this Chapter, we leverage the framework introduced in Chapter 5 to evaluate the generalization capabilities of QCBMs, presented for the first time in Ref. [33]. We demonstrate the QCBM's generalization performance as an integral component of its evaluation as a *good* generative model, and its ability to perform well with limited training data.

6.2 Experiment Details

For all numerical experiments, we train a 12-qubit QCBM circuit with a line topology to learn a cardinality-constrained target distribution $P_{\text{Target}}(x)$. This dataset has a solution space $\mathcal{S}$ defined by bitstrings that have a specific number k of 1s (e.g. '10001011' for $k = 4$). Thus the target distribution is uniform over the solution space, as shown by Equation 23.

The definition of such target distribution is only formal: in reality, it is not needed to have *a priori* knowledge of $|\mathcal{S}|$ for our metrics to be determined. The coverage metric (Equation 29) seems to require such knowledge. This requirement can be met when addressing benchmarking instances, such as the ones investigated in this work. However, in real-world cases, one is usually interested in comparing models (either different models or two different versions of the same model), hence one can simply compute their C ratios, which mitigates the fact that $|\mathcal{S}|$ is not known. Additionally, if one is aiming at estimating the coverage of a standalone model, one could simply use the asymptotic limit of the normalized coverage, given by Equation 36, which quantifies the total number of unique unseen and valid samples over the total number of queries and does not require knowledge of $|\mathcal{S}|$. This value would provide a sufficient estimation with regards to the number of unique values covered from the solution space that were not

seen during training. The important notion in this framework is having a target distribution which is uniform and whose support is given by samples in a well-defined valid sector. Although in the case of the cardinality-constrained data set $|\mathcal{S}|$ can be estimated exactly, there are several real-world instances where it is easy to determine whether a sample belongs to the valid space, but it is intractable to determine *a priori* the size of the support. Examples of this class of problems can be found in Appendix 6 of Garey and Johnson's comprehensive book on NP-Complete problems [113]. For example, the *zero-one integer programming* problem described as MP1 in Appendix 6 is an NP-Complete problem where it is easy to check whether a given sample satisfies the constraints, but where it is intractable to find the whole set of bitstrings which satisfy the constraints.

We note that for this specific study, it is important to have a large $|\mathcal{S}|$ for assessing generalization, so that we can have appropriately sized training sets when spanning over ϵ. As such, we choose $k = 6$ for all runs, where $|\mathcal{S}| = \binom{12}{6} = 924$. The circuits are trained via a gradient-free Covariance Matrix Adaptation Evolution Strategy (CMA-ES) optimizer [114] with a NLL loss function (defined in Equation 11) on the Qulacs quantum simulator [115]. While we optimize with NLL, we display the cost values in the form of the KL Divergence, as it is easier to visualize the success of the training process. Indeed, we know that for $P_{\text{Train}} = P_{\text{Model}}$, we have KL = 0. Each circuit is initialized randomly, and the maximum number of training iterations is $i_{\text{Max}} = 10,000$. Lastly, $10,000$ samples are generated for the generalization evaluation procedure.

To assess the QCBM's quality-based generalization in Section 6.4, we introduce a re-weighted *Evens* dataset, where the solution space $\mathcal{S}$ is defined by bitstrings containing an even number of 1s (e.g. '01010011'), and a cost is assigned to each sample that quantifies the sample's degree of *separation* γ. We define *separation* as the largest bit-separation between 1s in the bitstring (e.g. γ('11010001') = 4). As we would like to optimize for the samples with the lowest cost, we focus on the negative separation $c = -\gamma$. We utilize this cost to reweight the training distribution via a softmax function on the training set bitstrings, namely:

$$P_b(x) = \frac{\exp\left(-\beta c(x)\right)}{\sum_{i=1}^{|\mathcal{D}_{\text{Train}}|} \exp\left(-\beta c(x_i)\right)}. \tag{34}$$

Following Ref. [49], we set $\beta = \beta_1 \equiv 1/T$ where T is the standard deviation of the costs that is interpreted as a 'temperature constant' [70]. Note that by adjusting β, one can tune the degree of reweighting introduced

into the uniform distribution: for instance, choosing $\beta = \beta_2 \equiv 2/T$ increases the impact of the reweighting procedure. The artifact of reweighting the target distribution allows one to determine whether or not the model is able to learn this 'bias' induced by the bitstring costs in addition to the validity constraint.

The only additional change from the numerical experiments in Section 6.3 is that we use an all-to-all $L = 2$ ansatz rather than a line topology. In our numerical experiments, we found that using an all-to-all topology provided better validity-based generalization performances compared to the line topology. This observation is possibly related to a trainability issue of the latter topology at the number of layers needed to describe the reweighted dataset, since the uniform even distribution can be constructed exactly using a two-layered QCBM with a line topology. In general, for an arbitrary real-world dataset or more generic cases where one does not necessarily have an intuition of the type of circuit ansatz which might be suitable for the data, one needs to treat the exploration of the ansatz as an additional hyperparameter to be adjusted. Additionally, we use a fixed value of ϵ, with $\epsilon = 0.1$. As the *Evens* dataset is easier for the QCBM to learn, we use fewer layers and are thus able to avoid having too many parameters by implementing the all-to-all ansatz. We also remark that the choice of all-to-all compared to line topology on the *Evens* dataset improves trainability significantly even though the line topology is sufficient to represent this state.

Note that because the Evens solution space for the quality-based generalization assessment is much larger, 10% of the solution space accounts for a similar number of samples as used in the validity-based investigation. This might have helped in seeing a comparable performance with less percentage of data than the former dataset, although in reality the properties of each distribution to be learned can play a significant role and this data efficiency capability needs to be studied on a case-by-case basis. For the validity-based generalization demonstration in this work, we emphasize that the model is able to learn from a few training data, and as such, we span over ϵ intentionally. In a practical context, however, there is no need to know this percentage since all the metrics, quality and validity-based, can be computed without its knowledge.

6.3 Validity-Based Generalization

6.3.1 Expanding the Framework

As for this QCBM study, we are looking to vary the portion of training data , it becomes necessary to expand upon the framework introduced in Chapter 5. Note that when one is varying the size of the training set D, which can be controlled by increasing or decreasing the variable ϵ such that $D = \epsilon|\mathcal{S}|$, the metric computations may be individually affected as discussed below.

When computing the coverage metric across ϵ, we propose to use a normalized coverage $\tilde{C} = C/\overline{C}$, where $\overline{C}$ is taken to be the expected value of coverage, computed by:

$$\overline{C} = 1 - \left(1 - \frac{1}{|\mathcal{S}|(1-\epsilon)}\right)^{Q}. \tag{35}$$

This estimator, factoring in ϵ and the number of queries Q, indicates which coverage C one should expect when the generative model has perfectly learned the target distribution and generates samples accordingly [32]. When the number of queries is much smaller compared to the number of unseen solutions, i.e., $Q \ll |S|(1-\epsilon)$, we can approximate $\overline{C}$ as $\frac{Q}{|S|(1-\epsilon)}$. In this regime, the normalized coverage can be estimated as

$$\tilde{C} \approx \frac{|g_{\text{sol}}|}{Q}. \tag{36}$$

Interestingly, this expression does not need an explicit knowledge of $|S|$. We additionally propose a normalized rate $\tilde{R} = R/\overline{R}$, where:

$$\overline{R} = 1 - \epsilon. \tag{37}$$

The latter is the rate one should expect if the target distribution is learned perfectly. More specifically, the probability that a generated query is valid and unseen (i.e, rate) corresponds to the portion of the valid space that is unseen for a given ϵ, i.e., $(\mathcal{S} - D)/\mathcal{S}$.

For the fidelity F, we note that proposing a normalized F is non-trivial as this metric does not have an ideal value that depends on ϵ. If the target distribution is learned exactly, then we should see $F = 1$ independently of ϵ. But there is a non-trivial dependence on ϵ for any other model which is not perfect. To illustrate this dependence, imagine one obtains the same model after training under two different values of ϵ. These two models will yield, for example, the same values for the precision, since p does not depend on the size of the training set but only on the probability of a query being in the valid sector. Since a larger ϵ implies that the portion of seen

and valid data (the training set) is larger, this automatically implies that the sector of unseen and valid data is smaller and therefore the probability for a query to land there is smaller. Since the probability of landing in noise (unseen but invalid) remains the same, but the fraction of unseen and valid which appears in both the numerator and denominator of F changes, this yields different values for F for the same model. We leave the development of a normalized fidelity to future work, and highlight that this comment only becomes relevant when comparing across different ϵ values. When comparing models with the same ϵ, or assessing individual models, the metrics can be still used for accessing generalization.

6.3.2 Increasing Expressivity

We show an example of the validity-based generalization for QCBM models trained with various circuit depths $L \in \{2, 4, 8, 16\}$. We only ran experiments with an even number of layers, where, as explained before, every layer of single qubit gates is followed by a layer of entangling gates. For the expressivity study presented here, we utilize only 30% ($\epsilon = 0.3$) of the solution space in all runs, where the training data is uniformly sampled from all valid data (i.e., the solution space). In Figure 17, we demonstrate that this is an adequate value for seeing good generalization performance. We run 5 independent trainings of each model, and plot the result averages with error bars in Figure 16.

Across all results, we see that the deepest QCBM with $L = 16$ produces the best performance. Indeed, we see a direct correlation between increasing the number of layers, or the expressivity of the circuit, and a higher generalization performance quantified by the validity-based metrics. For shallower circuits with $L \in \{2, 4\}$, we barely see any increase in generalization performance across the average $(F, \tilde{R}, \tilde{C})$ values throughout training. Additionally, we see very little training in general as evident from looking at both the corresponding KL divergence, exploration, and precision trends. These models barely beat the random baseline [13]. After increasing the expressivity to $L \in \{8, 16\}$, we begin to see the model learn. The average $(F, \tilde{R}, \tilde{C})$ values steadily increase, where we see that the $L = 16$ model is able to reach the average values $(0.65, 0.67, 0.92)$, thus achieving high quality performance well above the average random baseline $(0.17, 0.23, 0.91)$. These metrics clearly show that the $L = 16$ QCBM is producing more unseen and valid samples with each

[13] The random baseline for each metric is computed from samples randomly drawn from the set of 2^N possible bitstrings [32].

new iteration step, and is learning the valid samples, distinguishing them from the noise. At the same time, the model's total exploration decreases as expected, as it begins to produce both valid samples from inside and outside of the training set. Table 17 displays individual values for the different metrics and Table 18 includes values for the random baseline.

While not always an attainable metric, due to the fact that the target distribution is typically unknown, one can compute the target KL Divergence $\mathrm{KL}_{\mathrm{Target}} = \mathrm{KL}(P_{\mathrm{Target}}||P_{\mathrm{Model}})$ and compare it to the usual KL Divergence, relative to the training set $\mathrm{KL}_{\mathrm{Train}} = \mathrm{KL}(P_{\mathrm{Train}}||P_{\mathrm{Model}})$, in order to see if the QCBM's output distribution is closer to the target than the training distribution. We see in Figure 16 that for $L = 16$, we have $\mathrm{KL}_{\mathrm{Target}}$ is smaller compared to $\mathrm{KL}_{\mathrm{Train}}$, indicating that the model is not overfitting to the training set. Note that we take the definition of *overfitting* to be *memorization* of the training distribution. We highlight that the values for $\mathrm{KL}_{\mathrm{Train}}$ are quite high compared to $\mathrm{KL}_{\mathrm{Target}}$, and according to this metric alone we could infer that the model does not have good generalization. However, displaying this metric alongside the $(F, \tilde{R}, \tilde{C})$ values, we are able to see that good generalization performance occurs even when $\mathrm{KL}_{\mathrm{Train}}$ is higher than zero. These numerics further support that achieving a $\mathrm{KL}_{\mathrm{Train}} = 0$ does not necessarily mean the model is producing good generalization performance: this metric alone should not be utilized to assess a generative model unless one is interested only in its data-copying capability.

We make a similar argument with the precision p and the exploration E. While simply providing p or E would not be enough to quantify the model's generalization performance, observing these values alongside the $(F, \tilde{R}, \tilde{C})$ metrics give us a broader picture. As the fidelity increases, despite the exploration decreasing, we can infer that this is simply because the model is disregarding noisy unseen samples. Additionally, as the rate increases alongside increasing precision, we can infer that a decent-sized portion of the valid samples being generated are unseen.

Interestingly, circuits with both two and four layers produce a similar $\mathrm{KL}_{\mathrm{Target}}$, and even though the 2-layers circuit displays a higher coverage, it contains more noise (less rate and fidelity than the 4-layers circuit). Although the models happen to have roughly the same $\mathrm{KL}_{\mathrm{Target}}$ value, our metrics allow one to disentangle the different generalization contributions and provide insights into the strengths and weaknesses of the models. This example also shows how judging performance just based on $\mathrm{KL}_{\mathrm{Train}}$ can be very misleading (with the 4-layers circuit largely outperforming the 2-layers

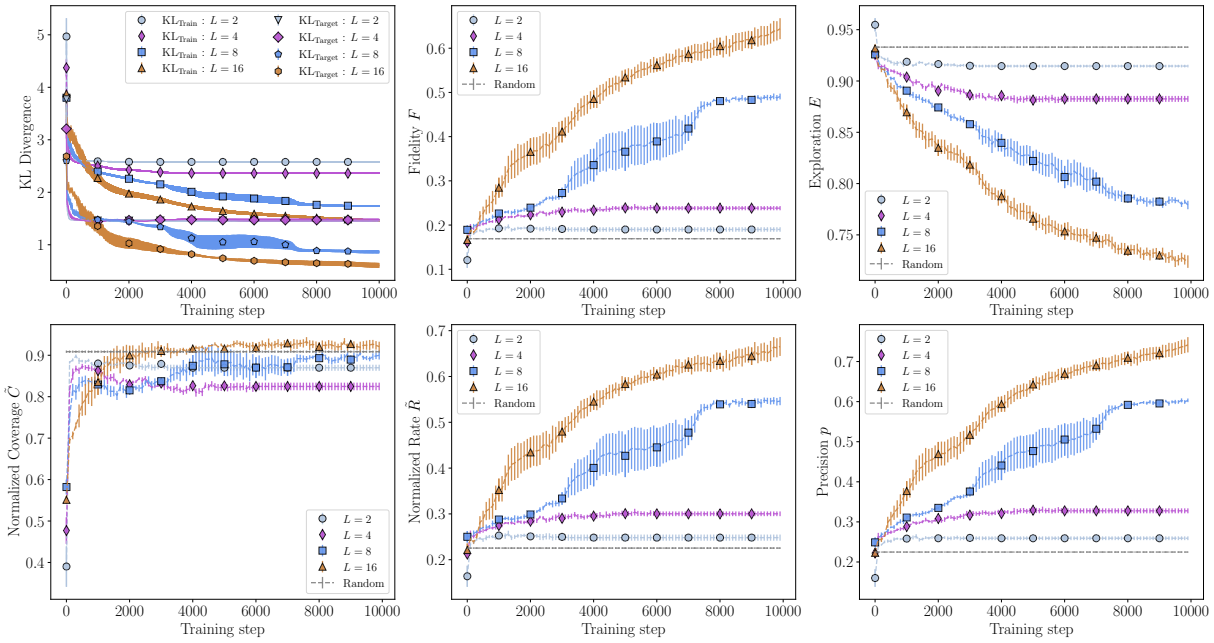


Fig. 16: **The QCBM's validity-based generalization performance throughout training across various circuit depths.** For each model with a different circuit depth $L \in \{2, 4, 8, 16\}$ and for $\epsilon = 0.3$, we show various metrics per training iteration including: the KL divergence of the model distribution relative to both the training and the target distribution (top left), the fidelity F (top middle), the exploration E (top right), the normalized coverage $\tilde{C}$ (bottom left), the normalized rate $\tilde{R}$ (bottom middle), and the precision p (bottom right). Note that these are average values over 5 independent trainings, where the error bars are computed by $\sigma/\sqrt{5}$. We see that the generalization performance increases throughout training, and that while for the majority of metrics all models are able to beat the random search baseline, the generalization performance is best for the deepest circuit model.

one), while the generalization metrics are more faithful to the $\mathrm{KL}_{\mathrm{Target}}$.

6.3.3 Reducing the Amount of Training Data

We investigate the effect of the size of the training set on the model's ability to learn the valid correlations and generalize accordingly. We span over various portions of the solution space $\epsilon \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ to use in the training set. Figure 17 showcases the average generalization performance at the last training iteration across 5 independent trainings for circuit depths $L \in \{2, 4, 8, 16\}$. We see that the trend of increasing circuit depth for enhanced performance extends to multiple ϵ values, thus suggesting that having enough expressivity is crucial for generalization to occur in this data set. Independent of the number of layers, it seems that when $\epsilon = 0.1$, the amount of training data is too low for the model to properly learn. When $\epsilon = 0.9$, we enter the regime where there is too much training data, which leaves very little room for generalization. Overall, we see that the model achieves good performance with access to 30% of the solution space during training. Table 17 displays individual values for the

different metrics and Table 18 includes values for the random baseline.

For increasing ϵ, we see that the average exploration E decreases. Such behaviour is expected as we are artificially decreasing the number of unseen bitstrings that the model can generate by reducing the size of the unseen space. For the average $(F, \tilde{R}, \tilde{C})$ values, we note some interesting individual trends. F decreases after $\epsilon = 0.3$, whereas $\tilde{R}$ and $\tilde{C}$, and even the precision p, continue to increase slightly before decreasing at $\epsilon = 0.9$. We believe this discrepancy in the fidelity is related to the increase in the number of noisy unseen samples relative to the number of valid samples that the model could generate as we increase ϵ.

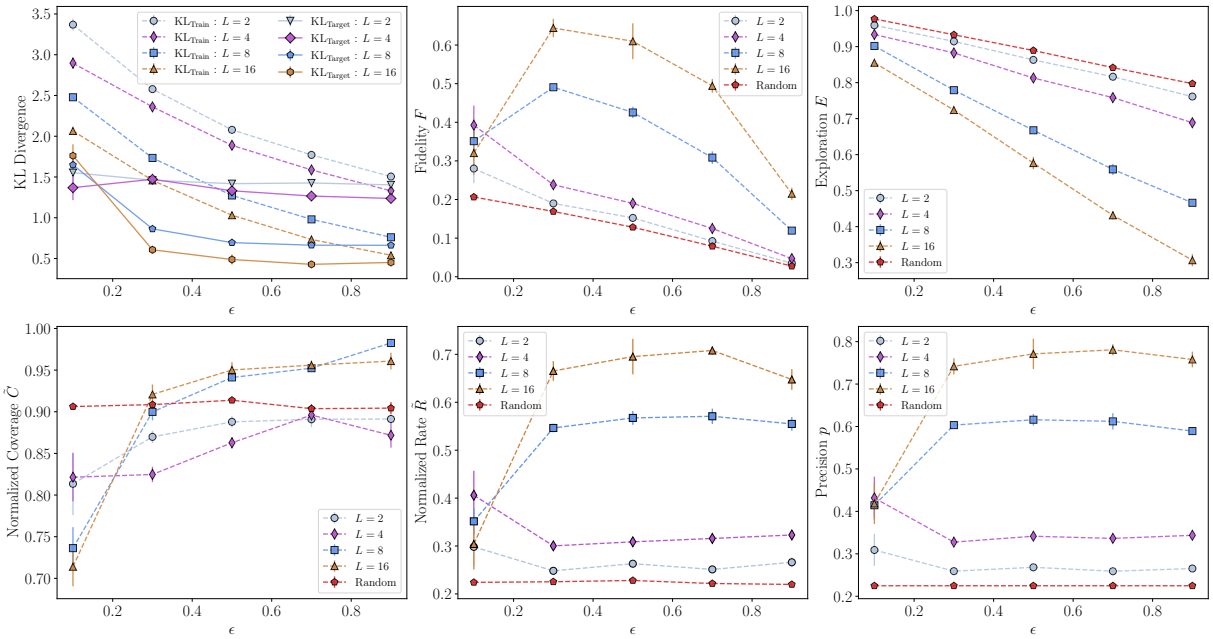


Fig. 17: **The QCBM's validity-based generalization performance on various training dataset sizes.** For each model with a different circuit depth $L \in \{2, 4, 8, 16\}$, we show various metrics across $\epsilon \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ values at the last iteration in training: the KL divergence of the model distribution relative to both the training and the target distribution (top left), the fidelity F (top middle), the exploration E (top right), the normalized coverage $\tilde{C}$ (bottom left), the normalized rate $\tilde{R}$ (bottom middle), and the precision p (bottom right). Note that these are average values over 5 independent trainings, where the error bars are computed as $\sigma/\sqrt{5}$. We see that the generalization performance increases for increasing circuit depth across all ϵ values. We see that for $L = 16$, with as few as 30% of the solution space used for training, the QCBM is able to exhibit great generalization performance on average: $(F, \tilde{R}, \tilde{C}) = (0.65, 0.67, 0.92)$, compared to that of the random baseline (in red): $(0.17, 0.23, 0.91)$.

The models achieve better performance than the random search baseline, with the only exception of coverage when the circuit depths are too low $(L = 2, 4)$. We see that $\mathrm{KL}_{\mathrm{Train}}$ tends to decrease with decreasing ϵ, and never encounters a turning point or a plateau. However, although $\mathrm{KL}_{\mathrm{Target}}$ presents the same decreasing trend until $\epsilon = 0.5$, it begins to plateau or change less dramatically until the largest value explored of $\epsilon = 0.9$. These two trends confirm that the QCBM model is indeed generalizing and not

memorizing the data set, as memorization would imply an increased value for $\mathrm{KL}_{\mathrm{Target}}$. The fact that $\mathrm{KL}_{\mathrm{Target}} < \mathrm{KL}_{\mathrm{Train}}$ is the expected behaviour for a model which is generalizing since this means that the learned model distribution is closer to the target distribution than to the distribution from the training set. The decreasing in the value of $\mathrm{KL}_{\mathrm{Train}}$ can be easily explained from this behaviour since the training set is closer as well to the target distribution for larger values of ϵ. This in turns explain the closing of the gap between the values of $\mathrm{KL}_{\mathrm{Train}}$ and $\mathrm{KL}_{\mathrm{Target}}$.

In summary, we demonstrate that QCBMs are able to go beyond memorizing a training distribution, and learn valid features in an underlying target distribution. They exhibit strong validity-based generalization across $(F, \tilde{R}, \tilde{C})$ values. However, we note that the QCBM requires deeper circuits in order to obtain good generalization performance, and this may pose a challenge for obtaining good results on near-term hardware.

6.4 Quality-Based Generalization

We assess the QCBM's quality-based generalization capability to see if the model is able to go beyond learning validity features in a dataset and learn an adequately reweighted version of it. As defined in Section 6.2, each bitstring in the *Evens* dataset's solution space is given an associated score for its negative separation $c = -\gamma$, such that we can create a re-weighted training distribution according to the softmax function defined in Equation 34. For 12 qubits, the negative separation cost is $c \in [-11, -1]$ with $c(\text{`}100000000001') = -11$ and $c(\text{`}111111111111') = -1$ as the limiting cases[14].

In Figure 18, we show the results of the typical behaviour, which we take to be the median value of the $(F, \tilde{R}, \tilde{C})$ results across 15 independent runs. Starting from the left, the first three distributions are the histograms of the valid samples generated after training the QCBM on the inset distributions (green). Across all $\epsilon = 0.1$ training sets (204 samples), the models do not see bitstrings with an associated $c < -7$. The distribution in the rightmost panel corresponds to 10,000 queries from the underlying uniform target distribution across all bitstrings in the *Evens* dataset mapped to their negative separation score. We share the average metric values across 15 independent runs in Table 19 and for the typical run in Table 20.

For all output distributions the precision and the validity-based

[14] When there is only one or no '1' in a bitstring, the cost is assigned to be zero. Those are only marginal cases, and they do not affect our analysis.

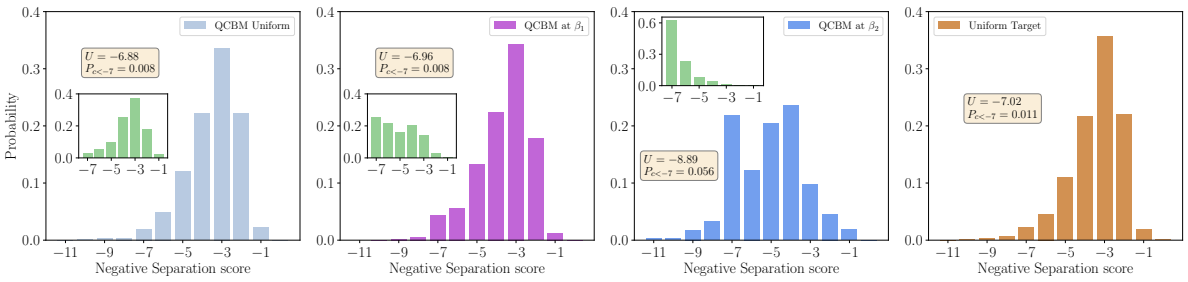


Fig. 18: **QCBM median output probability distributions when learning uniform vs. re-weighted datasets over** 15 **independent runs.** Each model is trained on a set of bitstrings in the *Evens* dataset, where each bitstring is mapped to its respective negative separation score, and to each training set a different degree of reweighting is applied. Note that we showcase the typical run across all models, as computed by the median $(F, \tilde{R}, \tilde{C})$ scores. We also note that the number of queries used to plot the histograms is 10000, except for the (green) histograms in the insets which correspond to the 204 samples used as the training sets ($\epsilon = 0.1$) for each degree of reweighting. Starting from the left, we show: the QCBM output when trained on a uniform training set; the QCBM output when trained on a re-weighted training set via a softmax function at inverse temperature $\beta_1 = 1/T$; the QCBM output when trained on a re-weighted training set via a softmax function at inverse temperature $\beta_2 = 2/T$; and 10,000 samples from the underlying uniform target distribution. While it is possible to observe values up to $c < -7$ when we sample 10,000 from the target distribution, across all training sets containing only 204 samples, the models do not "see" bitstrings with an associated $c < -7$. We highlight the quality-based generalization metric U for each distribution as well as the cumulative probability for generating samples where $P_{c<-7}$. With these two values, we are able to detect that the QCBM is able to achieve a lower U when we increase the level of reweighting in the training set, as well as a higher $P_{c<-7}$. Therefore, we see that the QCBM is able to effectively learn the 'reweighting bias' and generalize to data with lower negative separation costs than that of the training set.

generalization performance are very high, as supported by the results in Table 20. However, the distributions differ in their quality-based generalization capabilities, which we assess by computing the utility U metric described in Equation **??** and the cumulative probability for $c < -7$, denoted as $P_{c<-7}$. We introduce the latter metric as a method to ensure that we are not simply reaching the lowest scores due to over-sampling, as upon sampling enough, it is reasonable to assume that the model will reach bitstrings with scores $c < -7$. If the model's $P_{c<-7}$ is higher than that of the uniform target, we see this as a fair way to assess that the model has learned the 'reweighting bias' introduced in the training set.

When trained on a uniform training set, the model generalizes to the underlying uniform target distribution and generates some missing low cost bitstrings such that it has a $U = -6.88$ and a $P_{c<-7} = 0.008$ for the median run. [15] When trained with a re-weighted softmax training set, we see that the utility decreases to $U = -6.96$, while $P_{c<-7}$ stays the same. However, when we turn up the degree of reweighting in the softmax function by halving the temperature constant T, we see that the model

[15] We take the median run to be the median value out of all $F + \tilde{R} + \tilde{C}$ values, where we use the sum as a combined optimal score.

begins to learn beyond the validity-based features and on average generates more valid and unseen bistrings that have a lower associated cost than the minimum shown to the model through the training set. Remarkably, we see that the utility drops to $U = -8.89$ and the cumulative probability grows to $P_{c<-7} = 0.056$. These results support that the QCBM is able to effectively learn the 'reweighting bias' in the distribution, provided it is strong enough, and generalize to data outside of the training set with lower associated costs than that of the training set. This non-negligible quality-based generalization performance indicates that the QCBM may be useful for practical tasks in optimization, where one is interested in generating samples of minimal cost, by using them in the Generator-Enhanced Optimization (GEO) framework proposed in Ref. [49]. In future work, it would be beneficial to see if we can model higher-dimensional distributions with QCBMs for performing practical tasks with constrained optimization problems.

6.5 Summary & Implications

In the pursuit of obtaining a better fundamental understanding of whether or not QCBMs are good generative learners, it is imperative to investigate not only their data-copying capability, but also their learning capabilities via the assessment of generalization performance. The results here show that the QCBM exhibits good validity-based generalization performance when learning a desired feature in a cardinality-constrained dataset. Overall, we see generalization performance trade-offs when tuning various parameters. For instance, we show that one needs to increase the circuit depth, thus enhancing its expressivity, to obtain improved generalization performance. Additionally, we see that the model only requires access to 30% of the solution space to generalize well. We put forth these trends as an initial investigation into the QCBM's generalization capabilities. These results also beg the question: what is a *good* percentage of the solution space? This will be investigated in the future as we compare QCBMs with other models using this framework. As we scale up the circuit width, we believe these trends will serve as a good starting point for testing and comparing with other classical models.

We see that the QCBM's generalization performance is highly dependent on the number of valid data given to the model during training, and as such, it is of interest to dive deeper and conduct a more thorough future investigation into the QCBM's scarce data regime. In general, obtaining

an improved understanding on the selection of challenging distributions that QCBMs can learn well is an important part of future work. As we scale to larger quantum circuit models and aim to use QCBMs in a more practical sense, it is paramount to obtain a better understanding of what practical tasks could be handled well by QCBMs. The quality-based generalization results indicate that QCBMs are a good candidate for constrained optimization tasks, as the model can learn the correlations behind valid samples with associated low costs. In future work, it is important to scale up the size of the QCBM models and assess their capabilities on practical optimization tasks, such as portfolio optimization, where generalization capabilities have been shown to be a valuable asset [32, 49]. We foresee more elaborate training techniques for assessing the generalization performance of larger and deeper circuit sizes would be needed. This is one of the most fundamental challenges in quantum generative learning today, as will be discussed in more detail in Chapter 8.

As this study is the first formal assessment of QCBM's generalization performance, we hope our investigations will open the door for tackling future questions regarding the interplay of trainability [116, 117], expressibility [118], and generalization [32, 111, 31] in these models. In our study, we fail to see signs of over-parameterization, and in accordance with classical ML, we feel that this behavior might come to light if we continued to increase the number of layers. It remains an open research question to understand over-parameterization in quantum models, especially in the context of quantum generative models. For example, it would be interesting to see if phenomena such as double-descent which is present in deep learning models [119] due to over-parametrization happens in QCBMs as well. Additionally, we believe it will be important to investigate the generalization performance of QCBMs on quantum hardware, and investigate how noise impacts the training resources required.

Additionally, a consequence of this work is to motivate the scientific community to place importance on assessing the *generalization* over *memorization* capabilities when introducing a new quantum or classical generative model. We hope to see a shift in emphasis in the evaluation scheme towards prioritizing (or at least including) generalization performances.

7 HOW DO WE DESIGN GOOD QUANTUM GENERATIVE LEARNERS?

7.1 Background

As demonstrated in previous Chapters, learning complex probability distributions is a challenging problem for a classical computer, which is why improvements in quantum computing technology have prompted investigations into *quantum* generative modeling [21, 26, 20, 111, 28, 32, 120, 106, 105]. While QCBMs were the first algorithms for quantum generative modeling, they are not the only quantum generative learner that can be designed and investigated. QCBMs typically contain a very general ansatz with respect to the dataset to be learned. To design new quantum generative learners with intention, we can consider two interesting approaches [16]:

1. Classical-Inspired: The model contains a quantum circuit method to mimic a classical behavior that we know to be important for networks to learn complex datasets.
2. Hardware-Aware: The model takes advantage of novel complex functionalities that are being implemented in the latest quantum computing hardware - such as mid-circuit measurements and classical control [121, 122].

Classical-Inspired In the *classical-inspired* approach, one designs a quantum circuit that replicates some learning property or behavior that we see classically. Utilizing quantum building blocks for a desired behavior allows one to investigate the performance of this property directly and compare the resource complexity between the quantum circuit and the classical analogue that are designed to produce similar outcomes. One can then use this evidence to answer the question: *Is this classical generative learning property useful for designing quantum generative learners?*

Hardware-Aware In the *hardware-aware* approach, one designs a quantum circuit that intentionally uses complex functionalities that have been experimentally implemented in quantum computing devices. This approach allows one to take directly advantage of the *known* capabilities of quantum hardware, rather than elements from quantum theory that have not been experimentally realized. One can then use this evidence to answer

[16] Keep in mind that these are not the *only* approaches. This will be further discussed in Chapter 8

the question: *Is this quantum hardware functionality useful for designing quantum generative learners?* In addition, one can use algorithms of this type to benchmark and stress test the hardware capabilities.

In this Chapter, we explore the classical property of *non-linearity* in quantum generative learner, as shown in Refs. [34, 35]. To do this, we introduce the Quantum Neuron Born Machine (QNBM) - a quantum generative model that mimics the connectivity and non-linear activation of classical neural networks. With a significant amount of numerical, and some initial theoretical evidence, we first show the effects of introducing non-linearity into the state evolution of a quantum generative learner, and then we provide a deep investigation into the QNBM's potential to be a *good* quantum generative learner.

7.2 Non-Linearity in Quantum Generative Learners

7.2.1 Introducing the QNBM

One of the most fundamental differences between QCBMs and classical generative models is the presence (or absence) of linearity. Any classical neural-network-based generative model needs non-linear activation functions of neurons to learn non-trivial data [123, 124]. In contrast, QCBMs generate probability distributions through projective measurements of quantum states, which have been evolved in an entirely linear fashion. Therefore, it is interesting to see whether introducing non-linearities to the quantum state evolution can improve the quantum generative model's ability to learn challenging data distributions. Also, as new trapped-ion quantum computing devices have recently developed mid-circtuit measurements and classical control capabilities, it is useful to take advantage of these functionalities in a practical algorithm. As such, we introduce the Quantum Neuron Born Machine (QNBM) as a classical-inspired and hardware-aware quantum generative learner.

To do this, we expand upon previous work [82] that utilizes repeat-until-success (RUS) circuits [83, 84, 125] as quantum analogues of neurons. Building off of the quantum neuron discussed in Section 4.2.3, we discuss how connecting layer k to layer $k-1$ to design a feed-forward generative learner is a logical extension - one can repeat the QN circuit for every neuron in layer k (as shown in 19). Note that we restrict this work to $k = 3$, where the number of neurons in the input, hidden, and output layers are denoted as $(N_{in}, N_{hid}, N_{out})$, respectively. Also, notice that the ancilla qubit can be reused for every RUS circuit, since after every circuit

the ancilla is always in $|0_a\rangle$, and it is disentangled from all neurons. Thus, subsequent ancilla measurements do not affect already connected neurons. Likewise, connecting the *whole* network simply amounts to repeating the above for every pair of neighbouring layers.

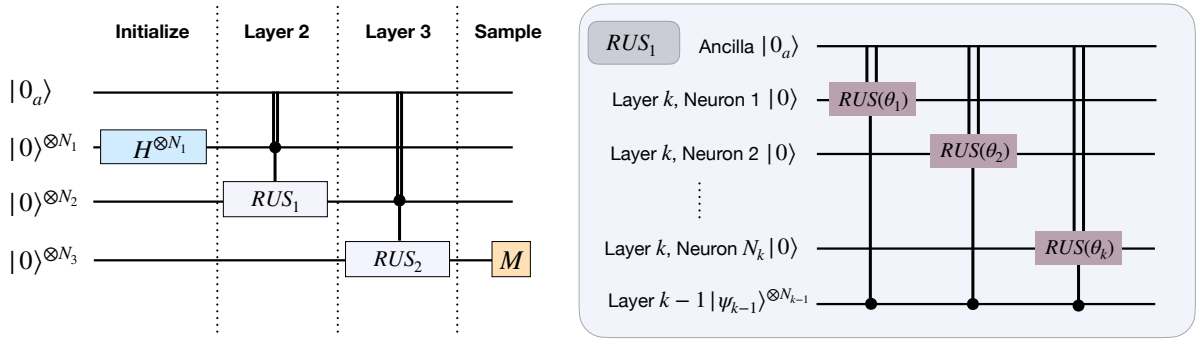


Fig. 19: **Quantum circuit component of the Quantum Neuron Born Machine** Left: QNBM circuit with 3 layers, where RUS1 and RUS2 represent connections between layers. The input neurons are initialized in a uniform superposition, and the output probability distribution is generated by measuring the output layer. Right: connecting two layers of a QNBM. This amounts to running RUS quantum neuron circuits for every neuron in layer k.

The quantum neural network proposed here is application-agnostic, and could, for example, be used in classification by adding data embedding and a cost calculation. To use it as a generative model, we make two alterations. First, the quantum state of the input layer: we initialize the first layer in a uniform superposition of bitstrings with Hadamard gates, as it is able to input some inherent structure into our circuit as an analogue to a classical prior distribution. Secondly, we only measure the qubits representing the output layer, in analogy to classical neural networks. The resulting probability distribution is then the output of the QNBM, and is optimized exactly the same way as in a QCBM (discussed in 4.2.2, with the weights and biases acting as the free parameters).

7.2.2 Variations & Scaling

It's interesting to consider the effect of non-linear activation not just in a binary way (is non-linearity useful?), but also in a quantitative way (is *more* non-linearity better?). It is, in fact, possible to alter the activation function and make it arbitrarily close to a step-like function, that acts as identity for $\theta < \pi/4$ and as a NOT gate for $\theta > \pi/4$ [82]. This can be done by recursively implementing the RUS circuit such that one produces a nested activation function

$$q^{\circ n}(\theta) = q(q(...(q(\theta)). \tag{38}$$

However, this comes at a cost of having n ancillas instead of one, as well $O(2^n)$ measurements and an increase in gates by a factor of $O(2^n)$. It is possible that the advantages of a more non-linear ($n > 1$) activation could outweigh the exponential resource requirements for larger datasets; however, in this paper we limit ourselves to a binary comparison of linear and non-linear models, and only use $n = 1$.

As a quantum circuit, the QNBM has $O(E)$ two-qubit parameterized gates, $O(N)$ one-qubit parameterized gates, and $O(N)$ two-qubit and one-qubit fixed gates, where E and N are the number of total edges and nodes (neurons) in the network, respectively. The circuit gets longer if any Quantum Neuron subroutines need to be repeated; [82] show, however, that the expected number of repetitions for each Quantum Neuron is bounded from above by 7, regardless of total network size.

Lastly, we note that the circuit also has $N+1$ (or in the case $n > 1$, $N+n$) qubits, of which only N_{out} neurons are measured to obtain the output. This results in a qubit overhead not present in other Born Machines. The larger total Hilbert space makes classical simulations of the QNBM longer than for a similarly-sized QCBM, but we do not expect the qubit overhead to increase the time resources of the model when realized experimentally.

7.2.3 Investigating Trends of Non-Linearity

To obtain a first understanding of our QNBM as a generative model, we assess the model's ability to express a target distribution that is defined by a cardinality constraint over discrete bitstrings. Given a search space of $2^{N_{out}}$ discrete bitstring samples, we define a sub-space $\mathcal{S}$ of valid samples x that have the desired cardinality $c = \lfloor \frac{N_{out}}{2} \rfloor$ (e.g. '0011', c = 2). The target distribution is uniform over these valid samples, and the goal of each QNBM is to generate high quality samples as a result of approximating the following distribution:

$$P_{\mathcal{S}}(x) = \frac{1}{|\mathcal{S}|}, \forall x \in \mathcal{S}. \tag{39}$$

We note that while we use these seemingly toy distributions to benchmark our small-scale QNBMs, learning larger cardinality-constrained distributions is shown to be a difficult and important task in combinatorial optimization, for instance problems with financial datasets [70, 32]. Each model is trained with a KL Divergence cost function, comparing the overlap of the target distribution $P(x)_{\text{Target}}$ and the generated distribution by the model $P(x)_{\text{Model}}$ during each training iteration. Post-training,

we assess the quality of the generated samples with the commonly used precision P metric. We take P to be the ratio of the number of valid samples Q_v to the total number of queries generated from the model Q. To demonstrate perfect learning of the target distribution, we should see the following behavior:

$$P = \frac{Q_v}{Q} \to 1. \tag{40}$$

We show results for architectures with $N_{out} \in \{2, 3, 4\}$, spanning over the number of neurons in the input and hidden layer. Note that we restrict the scope of these results to investigating one hidden layer, and leave a study of adding additional layers to future work. All simulations are run with an Adam optimizer with a learning rate $\alpha = 0.2$ and a step size $\epsilon = 0.1$ for a specified number of maximum iterations i_{Max}. For $N_{out} = 2$, we set $i_{\text{Max}} = 200$ and for all other simulations $N_{out} > 2$, we set $i_{\text{Max}} = 500$ as we need to increase the training time in order to accommodate for introducing more tunable parameters into the model. We initiate the QNBMs with uniformly random weights and biases from $(-1, 1)$. In order to demonstrate training robustness, we optimize each model over five different training seeds, and provide the average P values along with their standard deviations for $Q = 10^4$ generated model samples. These results are shown in Table 6. All simulations are run on IBMQ's Qiskit Aer Simulator with $N = 10^4$ shots.

Due to difficulties in simulating quantum circuits with classical control, we run each Quantum Neuron subroutine once (instead of repeating until success) and post-select experimental runs where every mid-circuit measurement produces the favorable outcome. To fairly quantify the model's time resources, we use the number of shots *before* post-selection as the appropriate measure in our circuit simulations and model comparison.

We display the performance of all the networks with $N_{out} = 4$ in Figure 20, where the networks are sorted by input and hidden layer size, as well as grouped into thirds by number of parameters: 8-12, 16-20, and 22-28. The data does not suggest significant trends from parameter number alone, but we do observe some trends when both parameter number and layer sizes are taken into account. Notably, the best performing networks have no hidden layer at all; the 3-0-4 and 4-0-4 achieved by far the best training, with over 95% precision and KL < 0.04. Both of these networks were in the top two thirds in terms of parameter number - this suggests that while the extra parameters from hidden layers appear to be redundant for QNBMs, one needs an input layer similarly sized to the output layer to

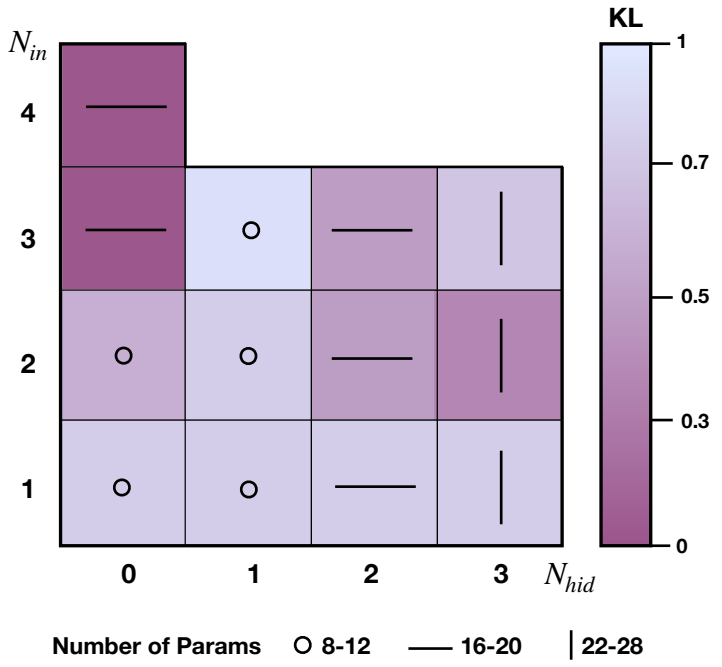


Fig. 20: **2D grid visualization of the KL Divergence vs. network layer size for a QNBM with a 4-dimensional output layer.** Each grid square corresponds to a specific number of input N_{in} and hidden layers N_{hid} in the network, where the color indicates the average KL cost value across 5 training seeds, such that the networks with darker squares reproduce the target distribution with a higher accuracy. We additionally highlight the number of tunable parameters associated with each network structure with a specific shape. The worst performance occurs when $N_{in} = 1$ (lightest shades), indicating that that the size of the input network is too small. However, we see that when increasing N_{in}, it is not necessarily advantageous to increase N_{hid} or simply the number of tunable parameters. Overall, we achieve the best performance when $N_{hid} = 0$ and for $N_{in} = N_{out}$ or $N_{in} = N_{out} - 1$.

sufficiently parameterize the model; this point is further strengthened by the $N_{out} < 4$ data left out from the plot, but shown in Table 6. We believe the claims about optimal network arrangement have sufficient evidence for small-scale models, but we acknowledge the possibility that hidden layers could become useful for large-scale problems.

7.2.4 Investigating Linear vs. Non-Linear

In addition to investigating the amount of non-linearity that exists in the network design, we study how *introducing* non-linearity into the quantum circuit model enhances the model's ability to learn a more complex, cardinality-constrained distribution. To achieve this, we compare our non-linear QNBM to a linear QCBM with the same $N_{out} = 5$, and a comparable number of tunable parameters: $P_{\text{num}} = 25$ for the QNBM with the topology $(4, 0, 5)$ and $P_{\text{num}} = 20$ for a QCBM with an all to all connected 1 single and 1 entangling layer. We select this QCBM topology as it has demonstrated great success in the literature [20]. Both models

$(N_{in}, N_{hid}, N_{out})$	P_{num}	N_{qubits}	**KL**	**Precision**
(1, 0, 2)	4	4	0.0041 ± 0.0001	0.9961 ± 0.0002
(1, 1, 2)	6	5	0.06 ± 0.02	0.94 ± 0.02
(2, 0, 2)	6	5	0.102 ± 0.002	0.903 ± 0.002
(1, 0, 3)	6	5	0.62 ± 0.01	0.59 ± 0.03
(1, 1, 3)	8	6	0.49 ± 0.03	0.66 ± 0.02
(1, 2, 3)	13	7	0.019 ± 0.004	0.984 ± 0.005
(2, 0, 3)	9	6	0.051 ± 0.002	0.951 ± 0.002
(2, 1, 3)	9	7	0.446 ± 0.003	0.665 ± 0.007
(2, 2, 3)	15	8	1.0 ± 1.0	0.6 ± 0.3
(3, 0, 3)	12	7	0.0517 ± 0.0008	0.946 ± 0.003
(1, 0, 4)	8	6	0.808 ± 0.009	0.460 ± 0.007
(1, 1, 4)	10	7	0.87 ± 0.05	0.44 ± 0.02
(1, 2, 4)	16	8	0.8 ± 0.1	0.49 ± 0.06
(1, 3, 4)	22	9	0.8 ± 0.2	0.53 ± 0.06
(2, 0, 4)	12	7	0.59 ± 0.03	0.64 ± 0.02
(2, 1, 4)	11	8	0.78 ± 0.04	0.45 ± 0.03
(2, 2, 4)	18	9	0.5 ± 0.1	0.56 ± 0.07
(2, 3, 4)	25	10	0.4 ± 0.2	0.7 ± 0.1
(3, 0, 4)	16	8	0.0373 ± 0.0005	0.960 ± 0.004
(3, 1, 4)	12	9	0.9 ± 0.1	0.49 ± 0.05
(3, 2, 4)	20	10	0.48 ± 0.09	0.63 ± 0.05
(3, 3, 4)	28	11	0.7 ± 0.1	0.5 ± 0.1
(4, 0, 4)	20	9	0.0389 ± 0.0002	0.959 ± 0.003

Tab. 6: **Average Precision and KL values for various QNBM network structures.** Here, we provide the average results with their associated standard deviations across 5 training seeded runs. For a given 3-layer QNBM network with the structure $(N_{in}, N_{hid}, N_{out})$, we provide the number of tunable parameters in the circuit, the number of qubits required, the output precision for 10^4 samples, and the KL divergence from the computed model output probability distribution. Ideal model performance is reached when we achieve $P = 1$ and $KL = 0$. For various N_{out}, we see that optimal structures occur when $N_{hid} = 0$ and $N_{in} = N_{out}, N_{out} - 1$, indicating that hidden layers may not be advantageous to the performance of the model at small-scales.

are trained with an Adam gradient-based optimizer with a learning rate $\alpha = 0.2$ and a step size $\epsilon = 0.1$ for the same number of iterations. We aim to keep the number of quantum and classical resources similar across both models to evaluate the models on equal ground to the best of our ability. We conduct 5 runs of each model with various training seeds, and use the best training for each model in our results, where we take the best training to be the model whose last loss is the lowest across the seed trials. All simulations are run on IBMQ's Qiskit Aer Simulator with $N_{\text{shots}} = 10^4$ shots.

We evaluate both models for their ability to produce two different probability distributions: one trivial distribution and one with added complexity. We take the trivial distribution to be one that is completely

uniform over the search space $\mathcal{U}$, such that:

$$P_{\mathcal{U}}(x) = \frac{1}{|\mathcal{U}|}, \forall x \in \mathcal{U}. \tag{41}$$

We take the second distribution to be the same as Equation 39 with $N_{out} = 5$ and $c = 2$, as introducing a cardinality constraint adds more complexity to the distribution by enhancing the difficulty of the learning problem. In Figure 21, we show the performance comparison of the QNBM to the QCBM on the two distributions by displaying their model output probability distributions produced at the end of training, and their KL training curves. We see that for the uniform target, both the QCBM and the QNBM are able to accurately model the probability distribution (QCBM: $KL = 0.0058$ QNBM: $KL = 0.0157$). Additionally, we see that throughout training, the QCBM takes fewer iterations to converge. However, when adding a constraint to the distribution and thus increasing the complexity, we see that the non-linear QNBM outperforms the linear QCBM with almost 3 times smaller error rate $1-P$ (QNBM: 0.05, QCBM: 0.14). In monitoring the loss throughout training, we see that the QCBM hits a plateau earlier on in training, while the QNBM is able to continue on in the training process and generate higher quality samples.

In addition to our comparison where we keep the allocated resources as similar as possible, we note that adding a second layer of single-qubit and two-qubit parameterized gates to the QCBM increases its performance. We display the probability histograms for the $N_{out} = 5$ QCBM with multiple layers, specifically 2 single qubit gate layers and 2 entangling layers of the topology mentioned in Section 4.2.2. In Figure 22, we compare these histograms with the target probability distributions for the uniformly distributed case and the cardinality constrained $c = 2$ case in order to detect if adding more expressibility to the circuit increases the training capabilities. As always, we use the Adam gradient-based optimizer with learning rate $\alpha = 0.2$ and step size $\epsilon = 0.1$, and we run the optimizer for 1000 iterations for the cardinality constrained distribution and for 200 iterations for the uniform distribution. While for the uniform distribution, we see that the model is able to learn the target distribution either just as well or better than the QCBM with 1 layer (single plus entangling), the cardinality constrained distribution becomes more difficult to learn with additional layers. In the case of the best training (the lowest KL across 5 seed training runs), we see that the model does not train well ($KL = 1.08$) and precision performance decreases ($P = 0.35$). Hence, increasing the expressibility does not help the QCBM outperform the QNBM for this

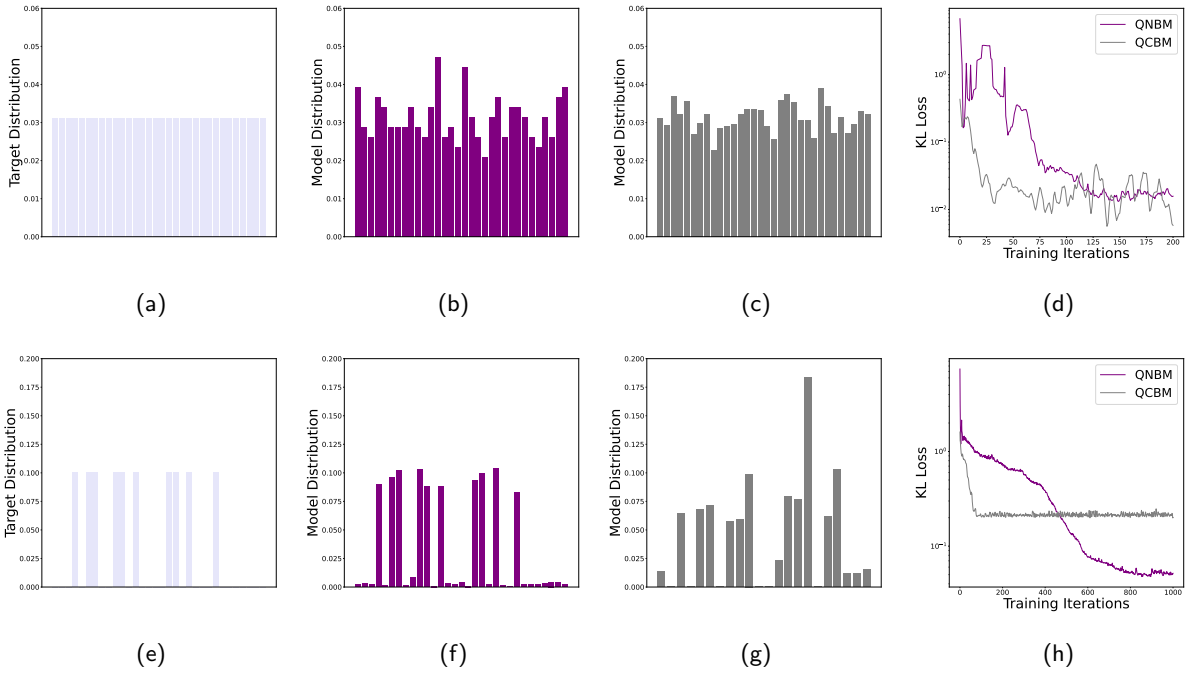


Fig. 21: **A visual comparison between the Non-Linear QNBM and Linear QCBM in modeling uniform and constrained target distributions.** All probability histograms contain ordered binary bitstring samples on the x-axis. (a) The target uniform distribution. (b) The QNBM model output with 10^4 samples. (c) The QCBM model output with 10^4 samples. (d) The KL divergence loss throughout training for the QNBM and the QCBM. (e) The cardinality constrained target distribution. (f) The QNBM model output with 10^4 samples. (g) The QCBM model output with 10^4 samples. (h) The KL Divergence loss throughout training for the QNBM and the QCBM. We see that the QCBM and the QNBM are both able to learn the trivial uniform distribution well, where the QCBM even outperforms the QNBM in achieving the lowest KL value in the fewest iteration steps. However, we see that with added complexity, the QNBM is able to outperform the QCBM and continue learning after the QCBM reaches its plateau in training, resulting in almost 3x smaller error rate.

task.

To further support that this training enhancement is due to non-linearity rather than just the neural network structure of the quantum circuit, we additionally train a "linearized" QNBM where Quantum Neuron subroutines are replaced with linear Pauli-Y rotations. The only effect of this is to substitute non-linear rotations $R_Y(2q(\theta))$ with linear rotations $R_Y(2\theta)$. We use the same cardinality-constrained distribution with $N_{out} = 5$ and $c = 2$ for the target. Again, we use the same optimizer, hyperparameters, and number of iterations as before. In Figure 23, we show that this model also struggles to learn the cardinality constrained distribution as well as the QNBM, suggesting that non-linear activation functions are essential to the good performance of the QNBM.

With this implementation, we are able to further support that it is not only the neuron structure that gives rise to the QNBM's good performance, but the combination of structure and the non-linearity. The model does not

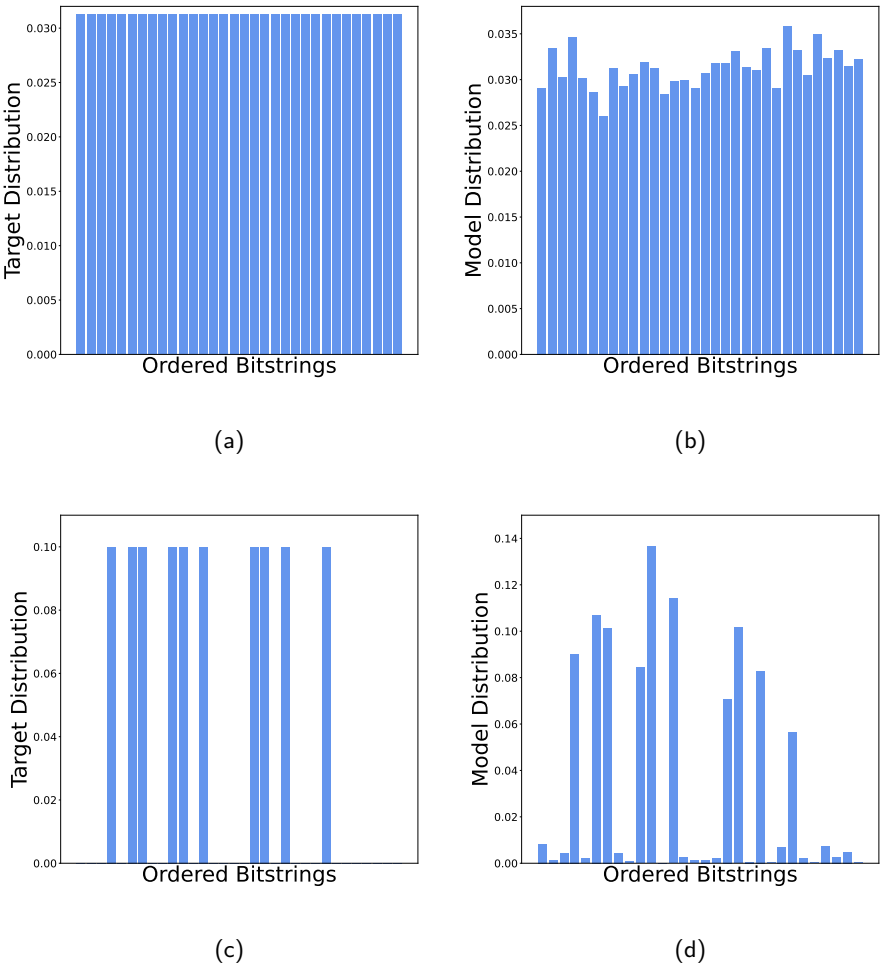


Fig. 22: **The QCBM model performance when given more expressivity.** (a) The uniform target probability distribution. (b) The 2-layer trained QCBM output probability distribution with $Q = 10k$ samples. (c) The cardinality constrained $c = 2$ target probability distribution. (d) The 2-layer trained QCBM output probability distribution with $Q = 10k$ samples. We see that for the uniform distribution, the QCBM is able to reproduce the target distribution well, but for the constrained distribution, the QCBM performs very poorly. Overall, the expressibility does not provide an enhancement for the QCBM to potentially outperform the QNBM.

train well at all, and has poor precision when sampled post-training. Over 5 seed runs, the best model achieves a $KL = 2.15$ with a corresponding precision of $P = 0.173$. We thus conclude that for this neuron structure, non-linear transformations into the state evolution is a requirement to achieve quality performance.

7.3 QNBMs for Hardware Evaluation

Part of the benefit of developing hardware-aware algorithms is the value of utilizing them to evaluate the most recent hardware capabilities. Quantum technologies have progressed rapidly over the past few years, including Quantinuum's recent advancement in quantum volume and features with

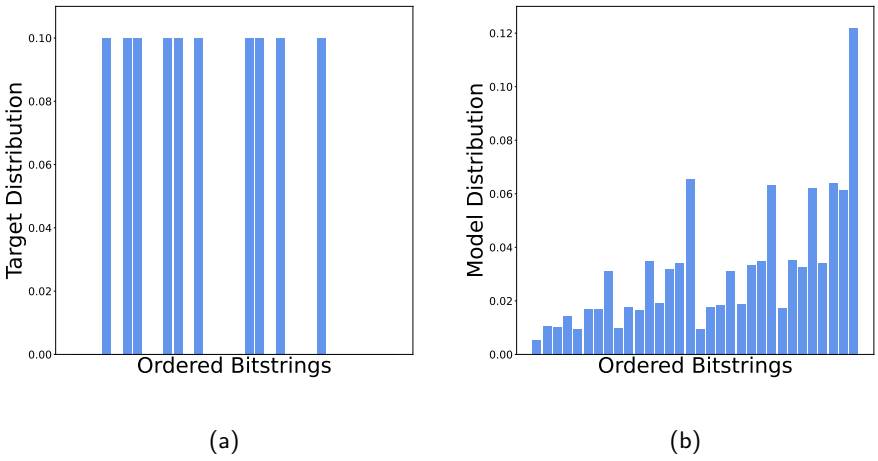


Fig. 23: **The linear QNBM model performance on the cardinality-constrained distribution.** (a) The cardinality constrained $c = 2$ target probability distribution. (b) The linear QNBM output probability distribution with $Q = 10k$ samples. We see that without the non-linearity, the model greatly suffers in training and learning the target distribution. Therefore, we consider the non-linearity to be a necessary resource in addition to the ansatz structure.

their H-series devices [126, 127, 121, 122, 128]. Alongside this progression comes the responsibility to challenge these machines with functionally complex algorithms that align with the new hardware capabilities. For example, NISQ algorithms [68, 69], while primarily developed to leverage limited quantum resources to perform classically challenging tasks, have also served as useful evaluation tools for present day quantum hardware [129, 105, 20, 130, 131, 132, 133]. As we now make the transition from NISQ towards fault tolerance [17] prior to , it remains useful to "stress test" - identify the non-obvious breaking points of - the hardware when running algorithms containing these more complex features at scale.

Stress Testing The intention behind *stress testing* is to assess a system by pushing it beyond its specified thresholds to identify the load under which it breaks down [134]. In software systems, stress tests can include an increase in concurrent virtual website users or a spike in client requests from a database. Stress testing can take on a slightly different interpretation when we consider stretching the load of a quantum system. In our case, the system to stress test is the Quantinuum H1 device, and we push the system by intentionally increasing the number of NISQ resources within the QNBM, an application-focused algorithm. In other words, using the QNBM model, we linearly scale the functional complexity by

[17] These transitional devices are capable of running algorithms containing non-trivial operations that are not typical of traditional NISQ algorithms like variational eigensolvers [**?**] and circuits designed for optimization [**?**]. The numeric quantity of a particular non-trivial operation (e.g classical control, mid-circuit measurements, etc. [122]) in the algorithm is the metric we utilize in this work for measuring functional complexity.

adding more RUS circuits (linear increase of input and output neurons) and then execute these various model sizes on the device. The total number of qubits required is equivalent to the total number of neurons, plus the ancilla. For each RUS circuit, there is an additional mid-circuit measurement of an ancilla, as well as an additional classical register to hold the value for a classically controlled operation. With each additional input-output neuron, there is an additional RUS circuit, where all other RUS circuits will have two additional parameterized two-qubit gates. For example, a $(2, 0, 3)$ contains 12 parameterized two-qubit gates, 3 fixed single qubit gates, and 3 fixed two-qubit gates. Whereas, a $(3, 0, 4)$ contains 24 parameterized two-qubit gates, 4 fixed single qubit gates, and 4 fixed two-qubit gates. Increasing the number of neurons also increases the number of measurement shot requirements as shown in Table 8. Thus, one can see that increasing the neuron size of the QNBM places an incrementally larger quantum gate, classical register, and measurement (mid-circuit and final shots) load for stress testing. We then assess if the hardware is able to execute a model of a particular size, how well it performs based on the algorithm output, and the point at which the system reaches its maximum capacity.

Quantum Computing Hardware Quantinuum uses the Quantum Charged-Coupled Device (QCCD) architecture for the H-series hardware [122], which allows for high gate fidelities and scaling. Within the device, ions are arranged in linear chains called *ionic crystals* confined on a surface trap. These crystals have the ability to be split, and their ions can be rearranged to execute entangling and single qubit gates on either a specific pair or individual ion, depending on the required operations. The ions can also be transported to other locations within the machine to minimize cross-talk when applying gates and measurements.

When a quantum circuit is submitted from a user to the device, qubits are assigned to physical ions such that the number of transport operations is minimized. The circuit is executed by conducting a series of transport and gating procedures, suitably pairing and isolating qubits for single-qubit and two-qubit gates, as well as measurements. Once all the operations are finished, the ions are restored to their original arrangement within the crystals, enabling the circuit to be repeated for the collection of measurement statistics. These results are then sent back to the user [122].

Due to the hardware's ability to spatially isolate ionic crystals, mid-circuit measurements as well as gate operations can be performed on the

target qubits with minimal risk of affecting the other ions, allowing for low cross-talk errors. The rearrangement and transport of ionic chains also allow for entangling operations on any pair of qubits, regardless of the spatial separation. This enables Quantinuum's machine to have all-to-all connectivity which reduces circuit depth and allows for high-fidelity computations.

Table 7 below provides insight into the current error and fidelity rates of Quantinuum's H1 machine.

Parameters	Min	Typ	Max
Single-qubit gate infidelity	1×10^{-5}	4×10^{-5}	3×10^{-4}
Two-qubit gate infidelity	1.7×10^{-3}	2×10^{-3}	5×10^{-3}
State preparation and measurement (SPAM) error	2×10^{-3}	3×10^{-3}	5×10^{-3}
Mid-circuit measurement and cross-talk error	5×10^{-6}	1×10^{-5}	2×10^{-4}

Tab. 7: **Quantinuum H1−1 Specifications**

Here, we specifically conduct a quantitative and qualitative performance evaluation of the Quantinuum H1-1 trapped ion device [128] with an interpretation of a Stress Test (used in software engineering)[134] driven by the QNBM algorithm. The QNBM algorithm can be tuned in functional complexity by scaling the algorithm's number of RUS routines. Using this proposed method, we (a) assess the hardware's capacity to manage a computationally intensive Quantum Machine Learning (QML) algorithm and (b) evaluate the hardware performance as the functional complexity of the algorithm is scaled. Alongside the quantitative performance results, we provide a qualitative discussion based on the insights obtained from conducting the stress test with the QNBM - i.e. we put forth the specific road blocks that appear when scaling the functional complexity of the algorithm.

7.3.1 Stress Test Quantitative Results

Target Distribution To investigate the QNBM's performance on hardware, we first train the model to express a target probability distribution P_{Target} of our choosing and then extract the trained parameters (weights and biases) that encode P_{Target}. For our case, we choose a cardinality-constrained target distribution. A *cardinality-constrained distribution* is one that assigns the same probability p to all bitstrings that contain a certain number c of binary digits equivalent to 1. The probability p is defined based on the set of bitstrings that fit this criteria,

$\mathcal{S}$:

$$p = \frac{1}{|S|} \tag{42}$$

To satisfy the constraint of all probability distributions $\sum_x p(x) = 1$, all other bitstring probabilities will be zero.

For our case, we choose the cardinality c to be 1 for all target probability distributions given to our models. For instance, the target distribution for a (1,0,2) QNBM neuron structure where $c = 1$ will be {'00': 0.0, '01': 0.5, '10': 0.5, '11': 0.0}.

We utilize this target distribution to first obtain the optimal weights and biases for the model structure, using a high quality training procedure on the device simulator. The simulated training losses are demonstrated in Appendix Fig. 41; one can see high quality performance - i.e. low final KL values - for the parameters chosen to run on the hardware. To ensure that the hardware results are evaluated fairly, we compare the output of the hardware to a new target distribution P'_{Target}, which is the model output distribution given from the simulated training. All target distributions that are compared to the hardware generated distributions are taken to be P'_{Target}.

Model Structure Given that we want to assess how the devices perform as we gradually scale the functional complexity of the algorithm - i.e. the size of the target distribution (number of output neurons) - we choose the following QNBM neuron structures: (1,0,2), (2,0,3), and (3,0,4). These specific structures contain no hidden layers, as networks without hidden layers have demonstrated the best training performance according to previous results in the literature [34].

We utilize 500 iterations to train all model structures, as these resource constraints were found to be sufficient for training QNBMs in Gili et al.[34].

Training Metric Each model size is then trained on a noiseless simulator using the Kullback–Leibler (KL) Divergence metric displayed in Equation 13 [18] which quantifies the distance between the target probability distribution P_{Target} and the distribution being trained P_{Model}.

The goal when training the QNBM model is to tune the weights and biases such that the distance between the distributions P_{Target} and P_{Model} converges to zero (KL $\approx$ 0). In other words, when the KL Divergence

[18] Due to the size of models we consider, the KL Divergence metric is sufficient. For larger models, we encourage the use of other probability distance measures such as Maximum Mean Discrepancy Loss.

reaches a near-zero value, the model has learned and can express the target distribution.

Resource Estimation and Training We utilize 500 iterations to train (1,0,2),(2,0,3) and (3,0,4) model sizes, as these resource constraints were found to be sufficient for training QNBMs in Ref. [34].

Using post-selection, we obtain the following shot requirements. Given that we have N_{out} output neurons, the model distribution (as well as the target) will be over $2^{N_{out}}$ possible bitstrings. Therefore, we need *at least* $N^{cc}_{shots} = 2^{N_{out}} * K$ samples from the model to obtain the full probability distribution, where K is an arbitrary constant. If we have N_{out} output neurons in our model, we have the same amount of RUS routines (given that there are no hidden layers). Each RUS circuit has $p_{N_i=success} = \frac{1}{2}$. Consequently, the probability that all N_{out} RUS subroutines are successful is $p_{succ} = \frac{1}{2^{N_{out}}}$.

Therefore, if the expected amount of samples needed from the QNBM is $2^{N_{out}} * K$ and the success probability of the entire model is $p_{succ} = \frac{1}{2^{N_{out}}}$, then the total amount of samples needed on the noiseless simulator using post-selection[19] is

$$N^{ps}_{shots} = (2^{N_{out}} * K)/p_{succ} \tag{43}$$

In Table 8, we show the number of shots required using post-selection rather than utilizing mid-circuit measurement capabilities.

	(1,0,2)	(2,0,3)	(3,0,4)
Classical Control	400	800	1600
Post-Selection	1600	6400	25600

Tab. 8: **Shot amounts required for each model size** depending on whether ones uses post-selection (N^{ps}_{shots}) or classical control (N^{cc}_{shots}).

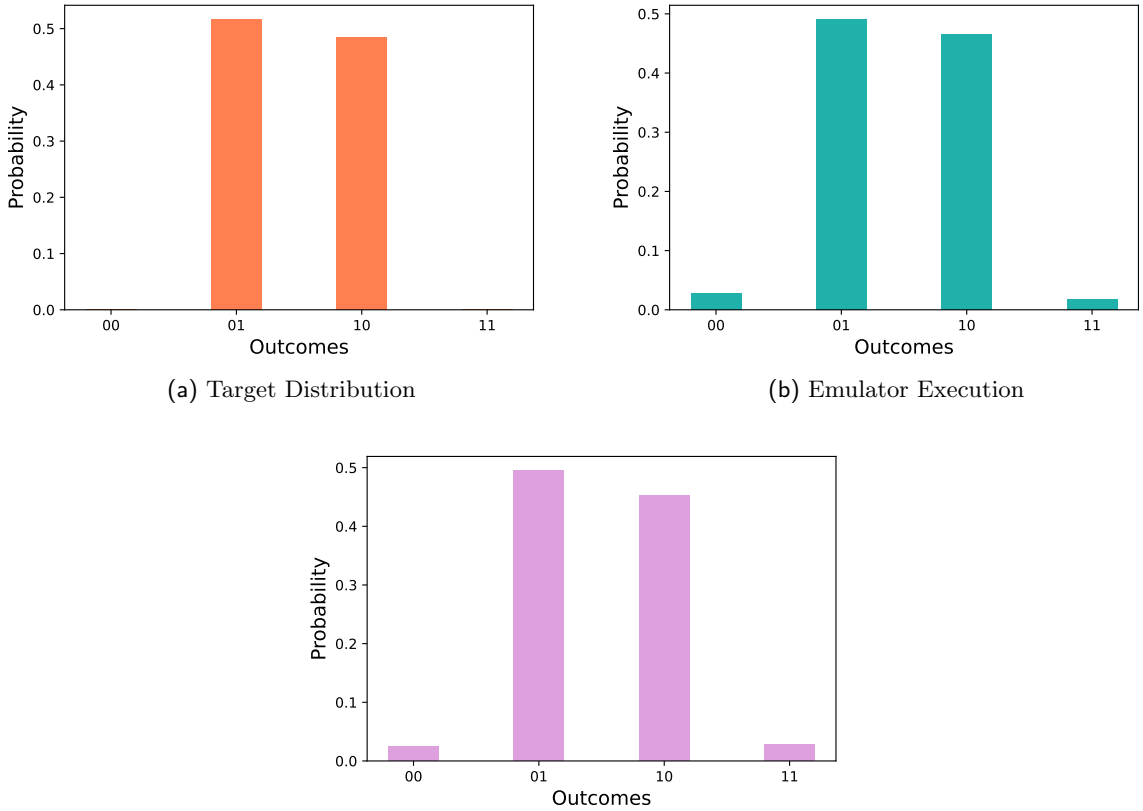
Performance Questions

1. How does the hardware perform when executing an algorithm of this complexity?

Here we demonstrate the high-quality results from the execution of the (1,0,2) QNBM model with the trained parameters on Quantinuum's H1 QPU and the QPU emulator, comparing both results with the target

[19] In our work, post-selection involves executing all RUS circuits (regardless of ancilla measurement result). Then, we only select the trials where all ancilla measurements were successful to collect statistics.

distribution P'_{Target}. As shown in Figure 24, the results from the hardware closely approximate the target distribution, reaching a KL value of 0.05.



(a) Target Distribution

(b) Emulator Execution

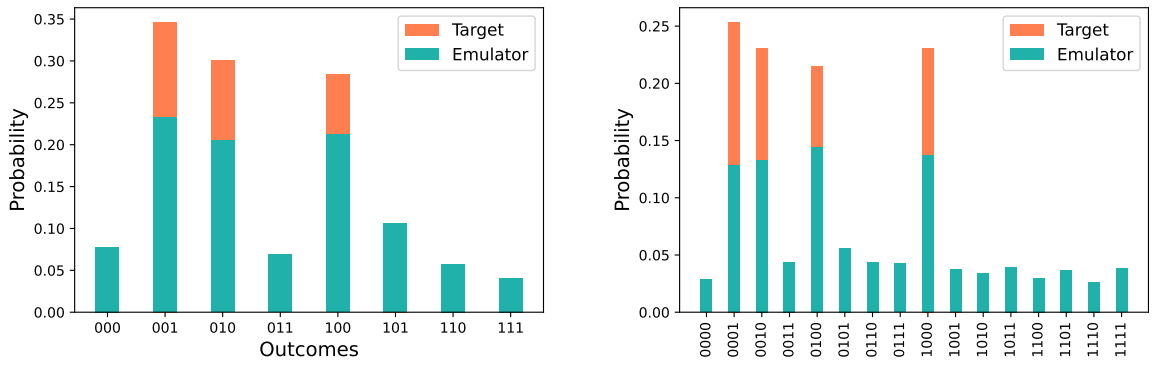
(c) Hardware Execution

Fig. 24: **Quantinuum's H-1 hardware and emulator results for a 2 Output neuron QNBM algorithm.** We show the (a) target probability distribution and (b) emulator and (c) hardware results from running on Quantinuum's H-1 series device. We see that both the emulator and the hardware achieve high accuracy relative to the target, and that the emulator and hardware are near identical.

This result highlights the exceptional performance of the algorithm on the hardware, specifically the novel classical control and mid-circuit measurement functionalities. Additionally, the results from the emulator and hardware have a massive overlap (KL = 0.0028). Therefore, due to scarce quantum and classical resources, we found it sufficient to execute the larger-scale models (2,0,3) and (3,0,4) on the emulator. We ran 8 independent trials on the emulator and chose the best performing model to display; the small standard deviation bars are present in 26. As shown in Figure 25, we can see that the generated probability distribution using classical control begins to diverge from the target distribution as the model size grows larger, and still, we see similar KL values for a $(2, 0, 3)$ and a $(3, 0, 4)$. For a $(2, 0, 3)$, we observe a KL value of 0.37 and for a $(3, 0, 4)$, we observe a KL value of 0.39. Even though these KL values are higher

than a $(1, 0, 2)$, one can easily identify the bitstrings that the model was executed to output.

Here, we show how the KL values of the target distribution P'_{Target} and the hardware/emulator output distribution scale as we increase the complexity of the algorithm via the number of output neurons. Again, we note that scaling the number of neurons directly translates to a linear increase in the number of classical control and mid-circuit measurement operations of the algorithm. Given Figure 26, we can see the linear leap in the KL values when scaling the number of output neurons. With this small set of data, we observe a linear trend-line of $y = 0.178x + 0.053$. It is difficult to make any concrete claims on scaling here because we have such a small set of data to analyze; although we can conclude that, as the model grows larger and despite the low noise levels in the hardware, the algorithm's performance does suffer. However, even with the higher KL values for these slightly larger models, we can still distinguish the cardinality constraint on the distribution by comparing the probability weights of the different bitstrings. For instance, in Fig 25b, one can see that the probabilities for the bitstrings 0001, 0010, 0100, and 1000 are significantly higher than the rest, despite the divergence from the target distribution. In future work, when the hardware is resource-ready for larger scale models, we can obtain a more clear trend in the decrease of the loss with respect to the number of neurons.



(a) 3 Output Neuron Emulator Execution

(b) 4 Output Neuron Emulator Execution

Fig. 25: **Emulator results for three and four output neuron QNBM algorithms.** Due to the similar performance of the emulator and hardware as well as limited quantum resources, we find it sufficient to execute the larger-scale models on the emulator. We demonstrate the output probability distributions (green) relative to the target (orange) when scaling the number of output neurons. The performance only worsens from a $(1, 0, 2)$ as we scale the functional complexity of the algorithm, and still, we are able to see with decent clarity the model output bitstrings that were relevant to the problem over noise.

2. How does the hardware perform when we scale the algorithm complexity

and add more RUS routines?

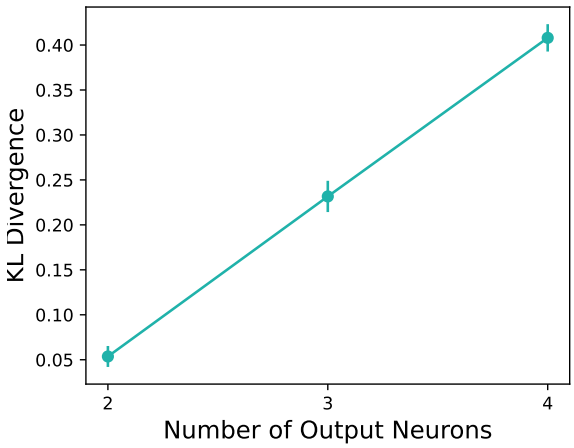


Fig. 26: **The relationship between the KL Divergence and increasing functional complexity of the algorithm (the number of neurons).** With three model sizes, we detect a linear trend in the data and see that the hardware's performance seems to moderately decrease according to the function $y = 0.178x + 0.053$ as we scale the number of mid-circuit measurement and classical control operations.

7.3.2 Stress Test Qualitative Results

Limited Amount of Classical Resources Within a quantum device, there are quantum registers that define the qubits, and there are classical registers used for holding runtime values such as integers or supporting conditional branches. We realized that, as our problem grew, the target machine H1 did not have enough classical registers to hold non-qubit values. Although counterintuitive, classical resources became a bottleneck for the quantum algorithm.

In prior literature, it has been shown that the average amount of executions of an RUS circuit needed for success is bounded above by seven [82]. However, due to the constraint of classical resources, we instead bound our Repeat-Until-Success algorithm above by six, giving the algorithm fewer branches and tries to succeed. Additionally, the resource constraint also placed a limit on the model sizes we could execute on the quantum hardware. Therefore, we found that executing a (3,0,4) model was the maximum size for the capacity of the current quantum device.

Cost Finally, executing on ion trap hardware is expensive and testing large-scale QNBM models beyond a (3,0,4) proved difficult. The costs for

each model size with the respective shot amount (Table 8) are shown in Table 9 as H-System Quantum Credits (HQCs).

	(1,0,2)	(2,0,3)	(3,0,4)
Classical Control	93	500	1570
Post-Selection	36	269	1822

Tab. 9: **Cost given in H-System Quantum Credits (HQCs)** to execute each model size depending on whether we perform post-selection or classical control. It is important to note how for smaller-scale models, post-selection seems more feasible and, as the model size grows, the benefit of classical control becomes apparent.

7.4 Is the QNBM a Good Generative Learner?

7.4.1 Investigating Trainability

In this section, we evaluate the QNBM's ability to effectively train on and express three target probability distributions typically utilized in the literature for benchmarking: Bars and Stripes (BAS), cardinality-constrained, and discrete Gaussian. Each distribution is defined on the set of 2^N bistrings. All simulations in our work are conducted with distributions of dimension $N = 6$, as this constitutes the size of each network's output layer. A BAS distribution is composed of uniform probabilities over bitstrings that represent either a bar or a stripe in a 2D binary grid. In this encoding, each binary digit represents a white (0) or black square (1) such that these patterns emerge [20]. For an $N = 6$ distribution with a 2×3 grid, there are 20 patterns for the model to learn [26]. A cardinality-constrained distribution contains uniform probabilities over bitstrings that fit a given numerical constraint in the number of binary digits equivalent to 1 (e.g. "001011" has a cardinality of 3). For a cardinality of $c = 3$, we have $\binom{6}{3} = 20$ patterns for the model to learn. Lastly, for the discrete Gaussian, we simply ask the model to learn the function $f(x) = \frac{1}{\sigma\sqrt{2\Pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$, where μ is the chosen mean and σ is the chosen standard deviation. For all simulations, the distributions are peaked at the central bitstring with $\sigma = 7$, as these values were appropriate for the distribution support size.

For this part of our investigation, we provide the model with complete access to the underlying probability distribution, rather than using a finite number of training samples. This method, prominently used in the literature to investigate alternative generative architectures such as Quantum Circuit Born Machines (QCBMs) [111, 20, 26], allows us to

investigate two important attributes regarding the model's ability to learn a wider range of distributions: the model's expressivity and its ability to be effectively trained. We want to emphasize that this is not a *generalization* investigation yet; however this is a start to understanding the model's potential to generalize.

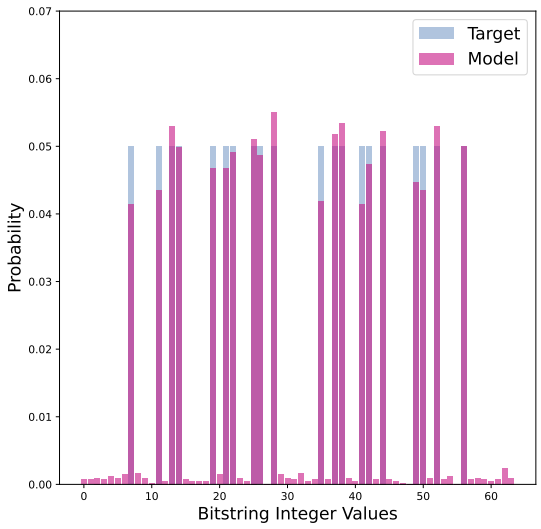


Fig. 27: **The QNBM training performance on the Cardinality Constrained distribution.** The QNBM is able to achieve a KL value of $KL = 0.04$.

The model's performance can be evaluated by computing the overlap of the model's output probability distribution $P_{\text{Model}}(x)$ with that of the target distribution $P_{\text{Target}}(x)$, using the KL Divergence shown in Equation 13.

Here, we provide the distribution learning performance of a $(N_{in}, N_{hid}, N_{out})$ $= (5, 0, 6)$ QNBM on the three distributions. It has been shown in previous work that QNBMs at small scales perform optimally when no hidden layers on introduced [34], and so we use this structure with small randomly initialized parameters. We train the QNBM with $2,000$ iterations and $10,000$ shots per iteration, for a total of 20 million shots throughout the training. When simulating the QNBM, we utilize postselection rather than implement mid-circuit measurements with classical control for each RUS sub-routine. The number of shots quoted is the number of shots before postselection. In practice, a quantum device running the QNBM would require the capacity for mid-circuit measurements and classical control to avoid an exponential shot cost.

We report the values of the best performing model across 5 independent trainings. The QNBM is able to achieve very low KL values with

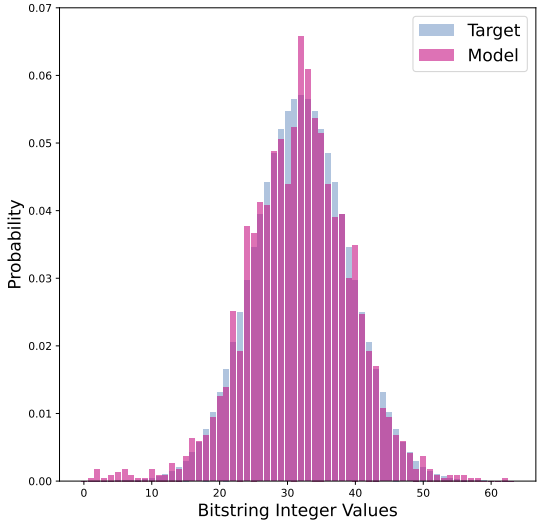


Fig. 28: **QNBM training performance on a discrete Gaussian distribution.** The QNBM is able to achieve a KL value of $KL = 0.019$.

$KL = 0.04$ for the cardinality-contrained distribution, $KL = 0.019$ for the discrete Gaussian distribution, and $KL = 0.099$ for BAS. In Figure 27 and Figure 29, we see the distribution outputs for each model trained on the cardinality-constrained and the BAS distributions respectively.

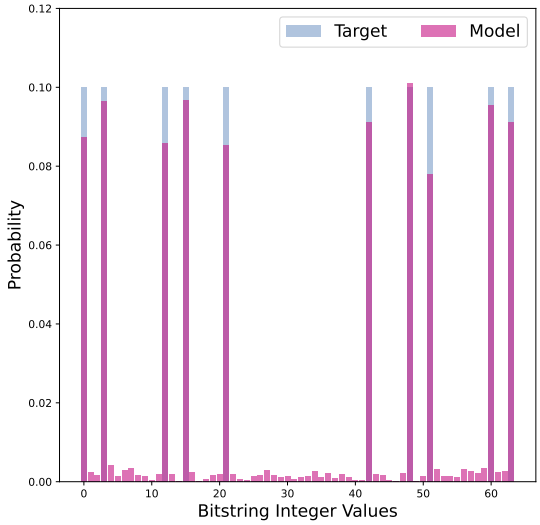
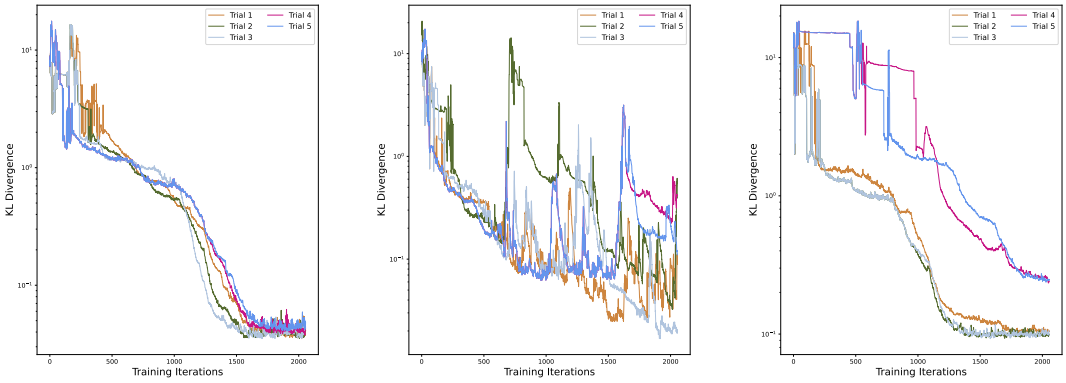


Fig. 29: **The QNBM training performance on the Bars and Stripes distribution.** The QNBM is able to achieve a KL value of $KL = 0.099$.

From these results we observe that the QNBM has high enough expressivity to represent these distributions and that it can be effectively trained to do so. Furthermore, we see that the QNBM is able to capture

representational patterns in discrete bitstrings very well. Thus, we have demonstrated initial evidence that the QNBM is decently expressive and capable of capturing two very important types of patterns: representational features within discrete bitstrings distributed uniformly and distribution shapes that arise from non-uniformity over a support of discrete bitstrings.

While the QNBM is able to achieve quality expressive performance across all distributions, it *does not* have the same stability in training on all three distributions. As shown in Figure 30a, the trainings are very stable, and that for the BAS distributions in Figure 30c, the simulations are stable in only 3/5 trials. For the discrete Gaussian in Figure 30b, the training stability varies greatly from the best performing to the worst performing model. It is unclear as to why the model is more unstable on this type of distribution, and we see value in investigating the training stability of this model as important future work, especially for understanding its scaling limitations and performance on other distributions.



(a) Cardinality Constrained (b) Discrete Gaussian (c) Bars and Stripes

Fig. 30: **The KL Divergence vs. the number of training iterations for the QNBM across independent trials for three distributions.** Here, we see that the QNBM is very stable on all trials of the cardinality-constrained distribution, while it becomes very unstable when training on the discrete Gaussian distribution.

Overall, the QNBM is able to achieve good expressivity, but can have volatile training performance when given access to the underlying distribution during training. Note that we *are not* using this evidence to claim that the QNBM is superior to all classical models, as we believe finding more theoretical bounds is necessary to make any such statements around quantum advantage.

7.4.2 Investigating Generalization

We take one step further in understanding the QNBM's capability to be a *good* generative learner by assessing its generalization performance, i.e. its ability to learn an underlying distribution $P_{\text{True}}(x)$ from a finite number of training samples. This is in contrast to the previous section, where all models were trained on the underlying distribution itself.

We assess generalization performance using an approach previously detailed in the literature [33, 135]. This approach is *different* to the one that is outlined in Chapter 5 and is more closely related to the one that is used in Figure 10 to provide additional support for the QCBM's ability to generalize. This is a better approximator of the model's potential in this context, as we are dealing with such small scale model sizes. For benchmarking models with samples dimensions $<< 12$ it becomes difficult to utilize the framework proposed in Chapter 5; there simply aren't enough non-training samples to take robust sample-based estimates. As such here, we do not take a sample-based approach, and compute the generalization based on computing the differences between exact probability distributions.

More specifically, this method consists of training the model on a finite number of samples from the underlying probability distribution and evaluating whether it can more closely approximate the true distribution from this limited training set. Concretely, at the end of the training, one can compare the overlap between the model and the training distribution P_{Train} with the overlap between the model and the true distribution P_{True}. If the model's encoded distribution is closer to the true distribution than to the training distribution, one can conclude that the model has indeed generalized to the true distribution from the training set. Concretely, the model has generalized if:

$$\begin{aligned} &KL(P_{\text{True}}(x), P_{\text{Model}}(x)) < \\ &\quad KL(P_{\text{Train}}(x), P_{\text{Model}}(x)) \end{aligned} \tag{44}$$

We will refer to the term on the left side of the inequality as KL_{True} and the term on the right side as KL_{Train}.

We probe the QNBM's ability to generalize samples from a superposition of three univariate Gaussian distributions. The three Gaussian distributions have means and variances (μ, σ) of $(8, 5)$, $(24, 12)$, and $(48, 7)$. We take 200 and 300 samples from the underlying distribution to construct two different training distributions and refer to these distributions as d_{200} and

d_{300} respectively. We show the underlying distribution as well as the two training distributions in Figure 31. Note that $KL(P_{\text{True}}(x), d_{200}) = 3.10$ and $KL(P_{\text{True}}(x), d_{300}) = 0.703$.

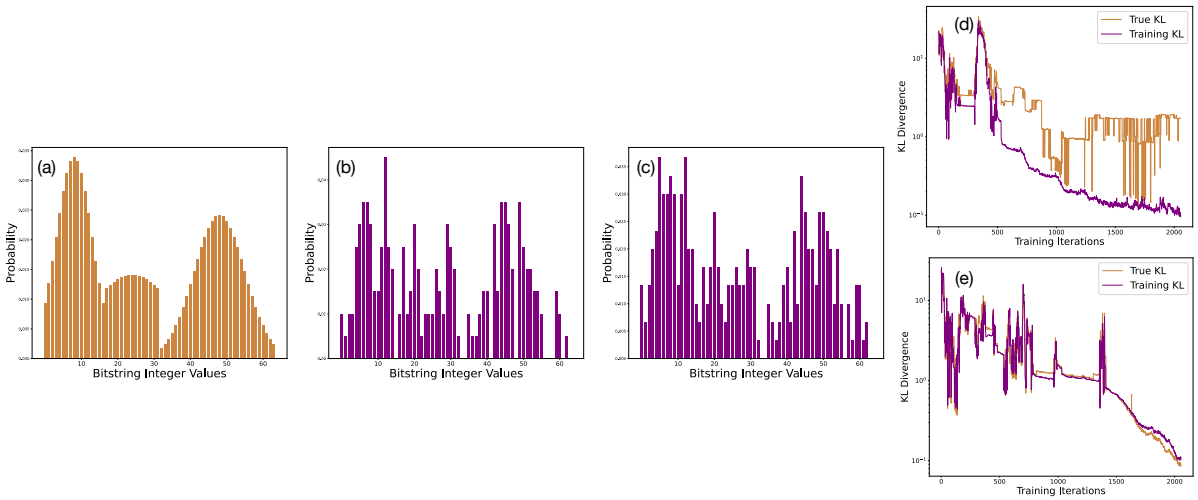


Fig. 31: **The QNBM's generalization performance on various training distributions sampled from the ground truth.** (a) The ground truth probability distribution $P_{\text{True}}(x)$. (b) A training distribution containing 200 samples from the true distributions, defined as d_{200}. (c) A training distribution containing 300 samples from the true distributions, defined as d_{300}. (d) The generalization performance for the training distribution d_{200}. (e) The generalization performance for the training distribution d_{300}. Throughout training, the model gets closer to both the ground truth and each training distribution, but for the training distribution d_{300}, the model's True KL (KL_{True}) is able to dip below the Training KL KL_{Train} more consistently, providing more concrete evidence for generalization.

We train the (5,0,6) QNBM with $2 \cdot 10^3$ iterations and 10^4 shots per iteration on each training distribution. We ran the QNBM on each training distribution over five independent trials with randomly initialized parameters. Since one typically does not have access to KL_{True} but can easily compute KL_{Train}, we report the performance of the model on the trial with the lowest final KL_{Train} and assess the generalization performance. We provide numerical values from all trials in Tables 10 and 11. In Figure 31, we showcase the model's training performance on this trial. The final KL_{Train} value is achieved by the model when training on d_{200} is 0.109, and the corresponding final KL_{True} value is 1.72. Since the final KL_{True} is much larger than the final KL_{Train}, we cannot conclude that the model generalizes to the underlying distribution in this instance. Since the KL between d_{200} and the true distribution is 3.10, it is likely that the training distribution is simply too different from the true distribution for the model to generalize. In other words, the model likely does not have enough information to approximate the ground truth well.

In contrast, the optimal final KL_{Train} value achieved by the model when training on d_{300} is 0.105, and the corresponding final KL_{True} value

Seed	KL_{Train}	KL_{True}
11	0.131	0.211
19	0.231	0.294
20	0.440	0.586
70	0.109	1.722
123	0.186	0.974

Tab. 10: **The final values of KL_{Train} and KL_{True} for each independent training after the QNBM trains on each training distribution.** The final values of KL_{Train} and KL_{True} after training on d_{200}. Note that the lowest value of KL_{True} ever attained is 0.211, and in this case we still do not have evidence for generalization as $KL_{\text{True}} > KL_{\text{Train}}$.

Seed	KL_{Train}	KL_{True}
11	0.239	0.454
19	0.105	0.085
20	0.560	0.553
70	0.359	0.353
123	0.206	0.176

Tab. 11: **The final values of KL_{Train} and KL_{True} after training on d_{300}.** Note that in four out of the five independent trainings, the model achieves $KL_{\text{True}} < KL_{\text{Train}}$, providing strong evidence that the QNBM is able to generalise the true distribution when training on d_{300}.

is 0.0853. The final KL_{True} value is less than the KL_{Train} value by a margin of 0.0197, indicating that the model is able to generalize to the underlying distribution. Since d_{300} contains more information about the true distribution than d_{200}, it is not surprising that the model demonstrates a stronger propensity for generalization when training on d_{300}.

This result suggests that the QNBM is indeed capable of generalizing. However, a more rigorous study with practical datasets is required to make stronger claims about the model's generative capabilities. In addition, rigorous generalization bounds are required to make any formal claims on quantum advantage for a particular distribution.

7.4.3 Investigating Implications of Deferred Measurement

The deferred measurement principle states that an unconditioned qubit can be measured at any point in the computation and it's probabilistic outcome will not change [36]. As the RUS non-linearity within the QNBM maps input qubits to the next layer of output qubits through a mid-circuit measurement protocol, it is possible to think that this mid-circuit measurement would provide one with information that can be used to simulate the probabilities classically. Here, we show that the RUS sub-routines *do not* allow us to trivially map this quantum model to a classical one, whereas a model without RUS sub-circuits containing mid-circuit

measurements could be mapped to a classical Bayesian network due to the deferred measurement principle of quantum mechanics.

Suppose we have a QNBM with n layers, where the first $k < n$ layers are connected, and we want to connect a neuron in the $k+1$'th layer to the network. The combined state of the first k layers, the qubit in the $k+1$'th layer, and the ancilla can be generally written as

$$|\psi_{k+1}\rangle = \left(\sum_{i=1}^{2^{N_k}} \alpha_i |\phi_i\rangle \otimes |x_i\rangle_k \right) \otimes |0\rangle_{k+1} \otimes |0\rangle_a , \tag{45}$$

where each $|\phi_i\rangle$ describes the first $k-1$ layers, and each $|x_i\rangle$ is a single bitstring state representing the k^{th} layer. Notice that if the network terminates at the k^{th} layer, then the probabilities of the various output bitstrings x_i are given by $|\alpha_i|^2$. By applying the gates as described in sec3:qnn, we get the pre-measurement state

$$\begin{aligned} |\psi_{k+1}\rangle = \sum_{i=1}^{2^{N_k}} \alpha_i |\phi_i\rangle \otimes |x_i\rangle_k \otimes (&\cos^2\theta_i |0\rangle_{k+1} \otimes |0\rangle_a \\ &+ \sin\theta_i \cos\theta_i |0\rangle_{k+1} \otimes |1\rangle_a \\ &+ \sin\theta_i \cos\theta_i |1\rangle_{k+1} \otimes |1\rangle_a \\ &+ \sin^2\theta_i |1\rangle_{k+1} \otimes |0\rangle_a), \end{aligned} \tag{46}$$

where $\theta_i = (\sum_{j=1}^{2^{N_k}} w_{ij} x_j) + b_i$. If we measure the ancilla and obtain result zero, then up to normalization the state becomes

$$\begin{aligned} |\psi_{k+1}\rangle = \sum_{i=1}^{2^{N_k}} \alpha_i |\phi_i\rangle \otimes |x_i\rangle_k \otimes (&\cos^2\theta_i |0\rangle_{k+1} \otimes |0\rangle_a \\ &+ \sin^2\theta_i |1\rangle_{k+1} \otimes |0\rangle_a). \end{aligned} \tag{47}$$

The probability of finding the connected neuron in the zero state and one state respectively become

$$P_0 = \frac{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 \cos^4\theta_i}{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 (\cos^4\theta_i + \sin^4\theta_i)}, \tag{48}$$

$$P_1 = \frac{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 \cos^4\theta_i}{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 (\cos^4\theta_i + \sin^4\theta_i)}. \tag{49}$$

Let us compare these expressions to those from a "linearized" QNBM, discussed in [34], where quantum neuron sub-routines are simply replaced with unitary Pauli-Y rotations. If we connect a neuron in the $k+1^{th}$ layer

to the k^{th} layer of neurons (starting from the same state as before), we obtain

$$|\psi_{k+1}\rangle^{(lin)} = \sum_{i=1}^{2^{N_k}} \alpha_i |\phi_i\rangle \otimes |x_i\rangle_k \otimes (\cos\theta_i |0\rangle_{k+1} + \sin\theta_i |1\rangle_{k+1}), \tag{50}$$

such that

$$P_0^{(lin)} = \frac{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 \cos^2\theta_i}{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 (\cos^2\theta_i + \sin^2\theta_i)} = \sum_{i=1}^{2^{N_k}} |\alpha_i|^2 \cos^2\theta_i, \tag{51}$$

$$P_1^{(lin)} = \frac{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 \sin^2\theta_i}{\sum_{i=1}^{2^{N_k}} |\alpha_i|^2 (\cos^2\theta_i + \sin^2\theta_i)} = \sum_{i=1}^{2^{N_k}} |\alpha_i|^2 \sin^2\theta_i. \tag{52}$$

Notice that the probabilities are *linear* combinations of the values $|\alpha_i|^2$ which also correspond to output bitstring probabilities if the network was terminated at layer k. Then the factors of $\cos^2\theta_i$ and $\sin^2\theta_i$ correspond to conditional probabilities. In other words,

$$P_{k+1}^j = \sum_{i=1}^{2^{N_k}} P(j|i)P(i) \tag{53}$$

where $P(j|i)$ means "probability to find a neuron in the $k+1^{th}$ layer in state j, given that the neurons in the k^{th} layer have been found in state i. The probabilities are $P(i) = |\alpha_i|^2$, $P(j|i) = \cos^2(\theta_i)$ for $j = 0$ and $P(j|i) = \sin^2(\theta_i)$ for $j = 1$.

Since we are able to write the probabilities $P_i^{(lin)}$ in this form, sampling bitstrings from the $k+1^{th}$ layer of the network is equivalent to the process of sampling bitstrings from the k^{th} layer, *classically* calculating the probabilities for the $k+1^{th}$ layer, and then sampling it from those. The calculation of probabilities and subsequent sampling requires $O(E_{k\to k+1})$ calculations, where $E_{k\to k+1}$ is the number of edges connecting layers k and $k+1$; therefore, such sampling is efficient. This is true for any pair of previously connected layers (Eq. 45 describes a completely general QNBM state), and as such, one can sample efficiently and classically from the whole network with $O(E)$ calculations, where E is the number of edges in the whole network.

In fact, this construction is precisely a Bayesian network [136]. Bayesian networks can be sampled from efficiently, and therefore the "linearized" QNBM can be efficiently sampled from as well.

Clearly the same construction does not work for the non-linear QNBM as the probabilities for the connected neuron are not linear combinations of the output probabilities for the layer that precedes it, and so there are no well-defined "conditional probabilities". While of course there may be other classical models equivalent to the QNBM, we believe the connection is certainly very non-trivial and not due to the principle of deferred measurement.

7.5 Summary & Implications

We feel that overall, our work with the QNBM provides further insight as to how non-linearity, a feature from classical ML, can be used to create a *potentially good* quantum generative learner. It additionally provides a framework for assessing more advanced quantum hardware. Still, some interesting questions remain for future investigation. While we now know that the mid-circuit measurement outcomes do not allow us to map the quantum model to a classical Bayesian one; we can investigate if those outcomes do provide us with any meaningful information that would help regularize the optimization. This is especially important if we see that non-linearity makes the training more unstable. Additionally, we could more theoretically investigate the inductive bias of the QNBM architecture and consider what datasets align well with its structure - i.e. what specific kinds of distributions is this model useful for? Is the non-linearity even necessary or does it only cause us more training pain at large scales?

Because the model is so structurally close to that of a Bayesian network, we find it interesting to compare the two. Do the learning problems that we typically use Bayesian networks for also align well with these quantum networks? Do the quantum networks exhibit any advantages that are detectable? As Bayesian networks are useful for modeling cognition, are these models also useful for for this task [137, 138]? Does the non-linearity introduced into quantum circuit structure allow us to model states of cognition?

Overall, we hope this work sheds more insight as to whether introducing non-linearity into quantum circuits for generative models is *useful*. While it's more clear that the model requires a quantum device, it is also now less clear that the model will be stable across general learning tasks at larger scales. Further evidence is required to make larger claims around whether this feature of classical ML is important for quantum architectures.

8 CONCLUSION: DO QUANTUM MODELS MAKE GOOD GENERATIVE LEARNERS?

As we continue to build quantum technologies that push the limits of computational power, it is important to consider the role that these devices could play in learning algorithms. In this thesis, we attempt to answer the very broad question: do quantum models make good generative learners? To do this, we showcase evidence from the Refs. [32, 33, 34, 35].

We first define *good* in the context of quantum generative learners, provide an evaluation framework, and demonstrate the approach on state-of-the-art models. This is important to challenge the idea that just because we have a quantum model that *fits* into the framework, it can produce results that are valuable. In the proposed framework, both classical and quantum generative models can be evaluated against other models for their quality generalization performance. We note that this is not the *only* quantitative method to call a quantum model *good*; it simply provides us with a road-map of *what we want* and how to evaluate it in a more practical, and less idealized data setting.

Next, we utilize the framework to benchmark the state-of-the-art quantum generative learner, the Quantum Circuit Born Machine, showing evidence that it has the *potential* to be a *good* generative learner for practical applications. Lastly, we put forth a novel generative model that is intentionally designed with a classically-inspired and hardware-aware approach, and provide a large amount of numerical evidence and some theoretical support that it has the *potential* to be a *good* generative learner. Note that we use the phrase *potential to be* in the above statements; this is because all of our evidence has been conducted at small-scale in comparison to the size of classical generative networks today. At small-scale, quantum models *are* good generative learners. However, for them to be practically useful with large amounts of data, or outperform classical networks in some meaningful way, we either need to discover a method to efficiently train large scale networks, or uncover where the specific, small-scale models might be valuable.

Other than obvious access limits to high fidelity, large capacity quantum computers, trainability of *general* large scale quantum models remains one of the biggest challenges in the QML field. This is a challenge that specifically exists for models such as QCBMs, where the ansatz does not contain an intentional inductive bias towards the dataset. We know high expressivity can lead to barren plateau phenomena, where gradients of the

cost exponentially vanish [116]. Emphasizing quantum learning settings with trainability guarantees, is a prominent area of research in the QML community. In this setting, one can ask the question: *For a given initial circuit design and optimization method, is the model efficiently trainable?*. There have been many insights from from a trainability perspective that remain highly useful [116, 118, 139, 117, 140]. For example, we know that our current methods for training general quantum models will not be more efficient than classical backpropogation [141]. We also know that models with pre-trained parameters and local costs can avoid BP phenomenon [72, 142]. Lastly, there is research focused around the development and benchmarking of new, efficient optimization schemes for general quantum learners.

Outside of trainability-focused research, we classify alternative *intentional* approaches into three categories as displayed in Figure 32.

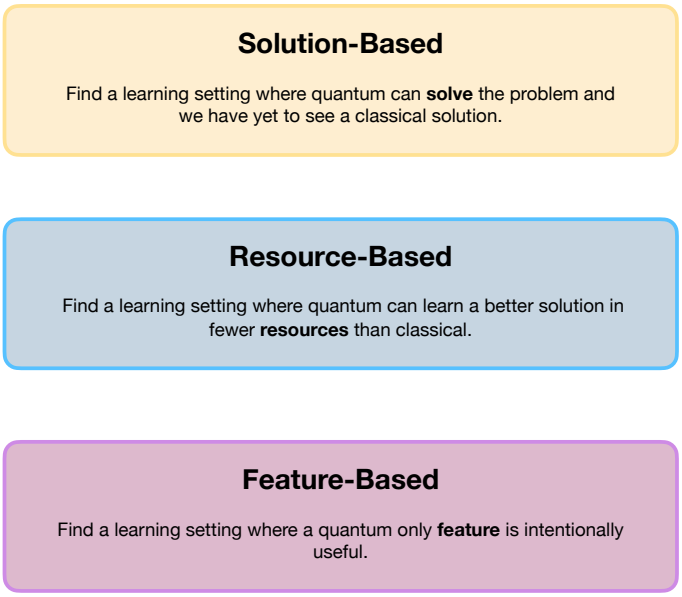


Fig. 32: **Three approaches to quantum generative research beyond trainability.**

It is not surprising that one of the most popular research approach of the last few years, borrows language and tools from computational complexity theory to look for quantum learning settings with a built-in computational speed-up [29, 31, 73, 143, 144, 145, 146]. This algorithmic quantum advantage approach focuses on demonstrating that a programmable quantum device can *exactly solve* a machine learning problem that no classical computer can solve in any feasible amount of time. We refer

to this as the *solution-oriented* approach, which requires one to find a problem that fits this requirement. A similar approach is *resource-oriented*, which typically attempts to find a learning setting where a quantum model can achieve better performance with fewer *resources* than classical. Here, resources can mean anything from amount of training data to the number of training iterations, or samples. One way to investigate this is to provide hard constraints for questions such as: *Can the model express and learn target distributions that are classically computationally hard to learn?* [29] and *Does the model exhibit a speed-up in sampling complexity for a specific target distribution?* [31, 73]. This method remains incredibly valuable for discovering a quantum model that can theoretically learn a class of target distributions outside of classical reach. However, it makes a lot of assumptions on the datasets and is does not predict trainability pitfalls in practice. Another way to tackle this resource-based approach is to ask questions that can be numerically benchmarked such as: *For a given practical task, can the quantum model generate more valuable out-of-training data more efficiently than the classical model?*. Answering this question often reveals insights regarding the model's trainability failures and specific resource needs (number of training iterations, circuit depth, etc.) for obtaining high-quality outputs. However, it typically lacks information regarding resource estimates once scaled.

Lastly, one can utilize a *feature-based* approach, where one aims to find a learning setting containing a quantum feature (one that classical systems do not have - e.g. superposition, entanglement, etc.) that is useful for learning. This prompts direct investigations of how quantum features influence learning; more specifically, we can address the potential utility that results from introducing a circuit model governed by the laws of quantum mechanics, into a learning problem.

In works that follow this thesis, we find it important to identify and construct models with inductive biases that arise from an element of quantum theory [147, 148]. This approach aims to increase our overall *understanding* of the potential natural alignments between quantum models and data structures that would lead to advantage. From this perspective, one may ask: *Does the quantum model contain an inductive bias that more strongly influences the model towards learning a particular structure in the dataset?*. In classical ML, we have seen models designed with an inductive bias toward the dataset outperform generic neural networks - supporting the claim that if you can bake known information into your model directly, that will exhibit an advantage [149, 150, 151, 152].

As we continue to investigate the role quantum plays in machine learning, we believe that it is essential that we understand the useful quantum specific *features* of quantum generative models.

We believe this body of evidence is overall *positive* towards quantum generative learners being labelled as *good*, and still we believe the results prompt further investigations into quantum generative learning. We hope going forward that other researchers are inspired to think outside of the box to consider ways in which we can investigate the relationships between quantum mechanics and generative learning.

9 ACKNOWLEDGEMENTS

The feeling of gratitude is *approximated* with words; gratitude as a word to communicate the feeling is the best thing it seems I can do. I am excited for the day that science and technology allows us to communicate our feelings less discretely.

First, I would like to recognize the Army Research Office (ARO) for providing funding through a QuaCGR PhD Fellowship. This work was financially supported by the U.S. Army Research Office (contract W911NF-20-1-0038) and UKRI (MR/S03238X/1). Not only did they invest in this research, but they invested in my personal growth as a scientist, writer, creator, mentor, philosopher, leader, and thoughtful human-being.

Second, I want to recognize my co-authors on the research publications that contributed to this thesis: Marta Mauri, Mykolas Sviestrys, Rohan Kumar, Aliza Siddiqui, Mohamed Hibat-Allah, Alejandro-Perdomo Ortiz, and Chris Ballance. Their scientific contributions were integral to the research presented in this thesis.

Next, I want to personally acknowledge the people in my life who I had the privilege to teach and learn from; the people who are one million times more important to me than any research contribution that I could make or degree I could obtain. Thank you Momma for encouraging me to be myself from day one; for learning to accept me, even when you disagree with me. Thank you Lisa for wearing so many hats in my life; for teaching me many moderately useful values and models that I have today - grace, authenticity, intentional spontaneity, and how to think for myself. Thank you Oana for your authentic friendship - from Marston walks to breakfasts at Gails to always making me a more aware person. Momma, Lisa, & Oana - your homes will always feel like home to me.

Thank you Dominika for all of the shared insights, laughs, goofyness,

chill-ness, and the mutually understood value of freedom. Thank you for being my favorite person to sleep in a car with, even when we know we both smell from hiking. Thank you Mykolas for being the most amazing workplace bestie; for making some of the most mundane tasks fun, and encouraging me not to work on days when I simply needed to feel. Thank you for kicking trees with me (not as bad as it sounds). Thank you Marta for inspiring me to be more considerate to myself, to take lunch breaks like a true Italian, and to focus more on doing our best than doing it quickly. Thank you Will-IAM for helping me become a more thoughtful and intentional person; for teaching me that just because I can, doesn't mean I should. Thank you for always challenging me to be better; for philosophizing and strategizing with me; for being someone I can balance power with. I appreciate the Line Street Boyz taking in a stray. Thank you Miggy for the shared connection of vulnerability; for being one of the most thoughtful men I have ever met; for the fun, openness, and balance you bring to my life. Thank you Dylan for sharing who you are with me and for supporting me in my journey to love my whole self - mind and body. Thank you Raghu for your incredible humor and invaluable support; for always checking in even when I was afar, for taking an interest in my career and personal well-being, and for taking off of work to cook Thanksgiving dinner with me. Thank you Melissa for making an effort to reconnect with me after years of lost connection. I appreciate that over the last 2-years, we've made an effort to learn about one another, and that we're still learning. Thank you Todd and Mimi for embracing me as a part of your family and for taking an interest in who I am. Thank you Steven for always embracing your love of teaching, and sharing that with me. Thank you Christie for always picking up the phone when I was in a crisis in a foreign country, even when it was 2AM your time.

Thank you Aliza for being exactly who you are and who you are going to be. Thank you for teaching me all that you do, and for extending my definition of gratitude beyond more than I ever thought. Thank you Jay for being an encouraging personal & professional mentor, and one of the best male role models I have ever had. Our conversations always give me something new to think about; I value that immensly. Thank you Ria & Rohan & Jamie for the opportunity to learn more about you, so that I can be a better mentor, supporter, and friend. Thank you Chris for being a research advisor that values freedom, and has encouraged me to pursue my own explorations - in both research and in life. Thank you David for encouraging me not to quit Oxford and for helping me pursue

my PhD in something I was truly passionate about. Thank you Alejandro for taking on an external research advisee/mentee; for your dedication to outputting quality research and for giving me opportunities to learn. Thank you Maria for being such a thoughtful and intentional internship research advisor; from being considerate and understanding to asking me insightful questions that encouraged me to think differently. I appreciate the values that you bring to research and the people you interact with; I want to lead like you do. Thank you Marcello for offering your time to challenge my work and help me become a better scientist. Thank you Denise for helping me obtain opportunities to mentor and connect with new inspiring people. Thank you Dr. Cruz for encouraging me to study Physics, even when I felt like I wasn't good enough. Thank you for telling a 16-year old me, that she was special and could change the world. And lastly, thank you James; even though you left my PhD journey early, you impacted the entire journey greatly. In truth, you helped put me on a personal journey that I am so grateful for. Wherever you are, thank you.

Again, thank you to every single person that contributed to my PhD journey, including those not listed here.

As always, a huge thank you to the environments that I got to become a part of as I conducted my PhD. I learned so much about societal structures, cultures, and traditions. Huge shout outs to Musty, Fasil, Sikose, and Sherry! I had the privilege of living in many places, so that I could finally understand for myself what *home* really means for me. As a result, this thesis was written from Boston, MA - and that's because I *learned* that for me, Boston is home.

The places: Oxford, UK; Berlin, Germany; Ljbluiana, Slovenia; Bratislava, Slovakia; Naples, Soverato, Rimini, Rome, Italy; Istanbul, Capaddoccia, Turkey; Jerusalem, Israel; Muscat, Oman; Santorini, Greece, Toronto, Canada; Romania; India et al; Warsaw, Poland; and CA, LA, FL, IL, CO, PA, MA, NJ, FL, NC, SC, USA. Thank you to all of the local people who became a part of the journey.

10 APPENDIX

10.1 Appendix Chapter 4

10.1.1 Training Details

Here, we provide additional details on the training process for both the quantum-inspired and the classical model. The TNBM, whose underlying

architecture is an MPS, is trained with a DMRG approach [19] with the negative log-likelihood cost function Equation 3, and the optimization is performed via Stochastic Gradient Descent with learning rate η = 1e-2. The number of parameters for the worst case in the TNBM is 1864 for our specific model of $\alpha = 7$. As the bond dimensions for each site are adjusted throughout training, we see that the TNBM does not reach the worst case, and instead has a total number of 1152 parameters. The total number of parameters can be calculated by summing over the squared bond dimensions at each site, and multiplying by a factor of 2.

In Figure 33, we show the training curves for TNBM with several values of the bond dimension α, reporting the KL divergence at each training epoch. Once more, we stress that we can detect trainability issues with our metrics that are confirmed by the learning curves trends. However, if we consider models that are successfully trained, we expect that our metrics should be able to detect the overfitting and underfitting regime when varying the hyperparameters (e.g. the bond dimension α which controls the TNBM expressivity).

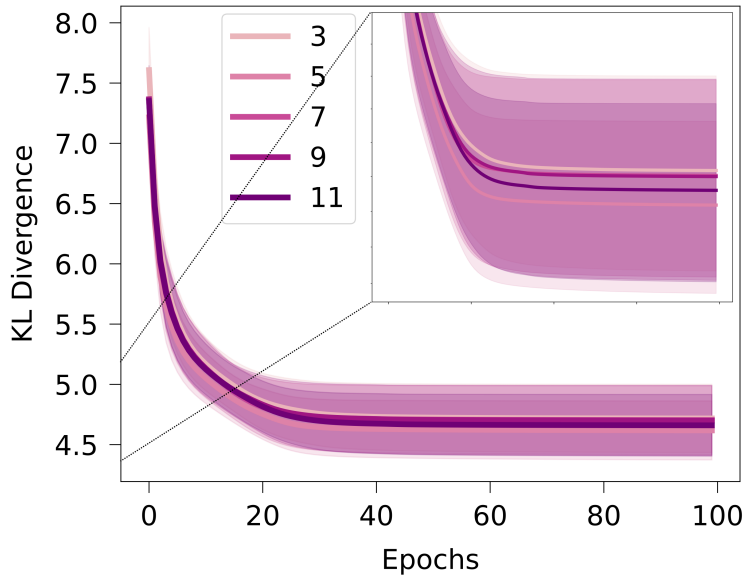


Fig. 33: **TNBM training curves for different bond dimensions.** We plot the KL divergence to monitor the training of the TNBM for bond dimensions $\alpha \in \{3, 5, 7, 9, 11\}$. The typical KL value is achieved for $\alpha = 7$ after 100 epochs, thus motivating our choice to utilize this value for further studies and model comparisons. The inset provides a more detailed view of the loss curves ordering.

In the case of the GAN, the architecture is set to be a feed-forward neural network with linear layers. The generator uses a Gaussian prior, ReLU activation function in the hidden layers and sigmoid cost function in the output layer. The discriminator uses Leaky ReLU activation function in all layers, along with a dropout operation before the final layer. The

optimization is performed via the Adam algorithm [153]. The values of the hyperparameters are shown in Table 2. The number of total parameters in the GAN is the sum of the parameters in the discriminator and the generator. For our specific architecture, the number of parameters is computed in each layer for the discriminator and generator, respectively. For our GAN with 1 hidden layer, we have a total of 4181 parameters.

10.1.2 Random Sampling

To better characterize the performance of the generative models under examination, we compare their generalization capabilities to a simple baseline: we sample randomly from the search space $\mathcal{U}$, thus collecting queries to compute the validity metrics, and we compare the results to the ones associated to the TNBM and the GAN. The metrics' values are summarized in Table 12: as expected, both generative models perform better than random sampling, which suggests that during the training process the models were indeed able to learn successfully, despite having different degrees of success. However, the coverage metric in the case of random sampling seems to be higher than the GAN, and this trend persists even considering different numbers of queries Q. What motivates this behaviour is the fact that the GAN suffers from mode collapse: its limited diversity impacts the coverage values, whereas the performance of random sampling is favoured by its higher diversity capabilities. However, Figure 34 shows that the GAN (Figure 34a) is able to generate more samples in the valid space or its vicinity than the random sampler (Figure 34b), thus explaining the higher fidelity of the former as opposed to the latter.

Metric	**TNBM**	**GAN**	**Random**
E	0.989(0.02%)	0.995(0.02%)	0.998(0.013%)
F	0.989(0.03%)	0.263(0.6%)	0.17(0.50%)
R	0.978(0.03%)	0.261(0.6%)	0.17(0.50%)
C	0.409(0.15%)	0.006(1.7%)	0.09(0.48%)

Tab. 12: **Pre-generalization and validity-based generalization metrics.** We display the average exploration E and the average (F, R, C) values for each best model run with an average and the associated relative percentage error across 15 query batches. Both the TNBM and the GAN achieve better performance than the random sampler for all the different metrics, except for the GAN coverage as pointed out in the main text.

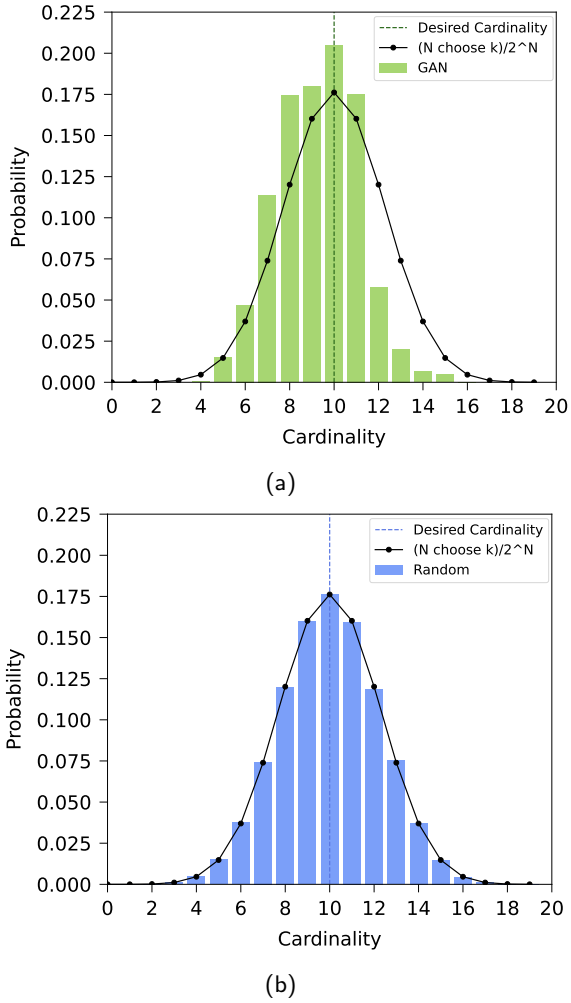


Fig. 34: **Cardinality distribution for GAN and random sampler.** The plots show the percentage of queries with different cardinalities generated by the GAN (Figure 34a) and by the random sampler (Figure 34b). We notice that the GAN is able to produce a higher number of queries with the correct cardinality $k = 10$ (or its vicinity), thus showing that the training process allowed the GAN to partially learn the validity pattern in the training dataset. The black line represents the probability to draw a query with a given cardinality when randomly sampling from the search space $\mathcal{U}$.

10.1.3 Metrics' Trends

To further demonstrate the power and stability of our metrics, we provide additional details regarding how they scale as we vary the number of queries Q generated from the trained model. Specifically, in Figures 35-38, we plot the values of the validity-based and quality-based generalization metrics and show that most of them do not change with the number of queries - except for coverage, as already shown in Figure 9, and for the minimum value that is displayed in Figure 37. The validity-based trend plots display the constant behaviour of the metrics for both TNBM and GAN as Q increases, along with a dashed black line indicating the ideal metrics value of 1. The quality-based trend plots display the constant behaviour of the

utility metric for both TNBM and GAN as Q increases, and a decreasing behaviour for the minimum value as Q increases. The latter is the expected trend: with more queries one has a higher probability of reaching a sample with a lower cost value. For both of these plots, we include a dashed black line indicating the training threshold. This data supports our claim that while our metrics are sample-based, most of them are not dependent on the number of queries.

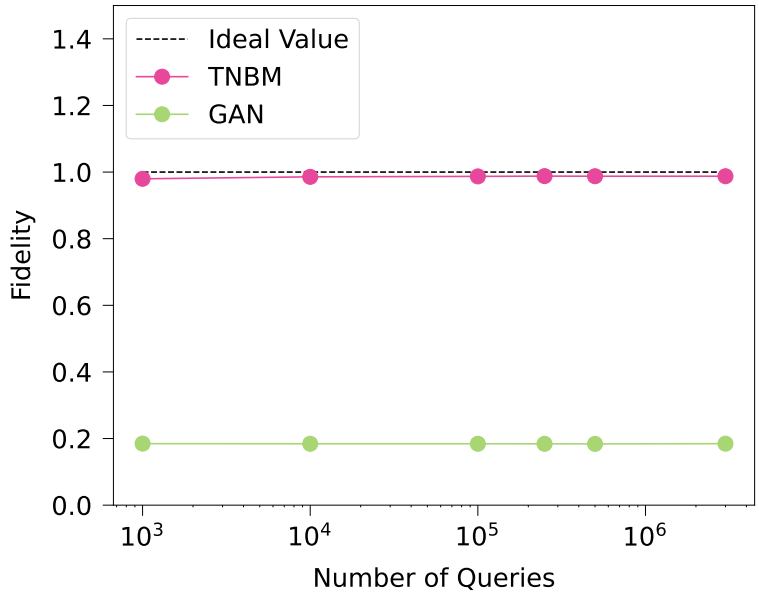


Fig. 35: **Fidelity trends for increasing number of queries.** The plot displays the constant behaviour of the fidelity F for both TNBM (pink) and GAN (green) as we increase the number of queries Q retrieved from the trained models. The dashed black line shows the ideal metric value of 1. In both models, the fidelity is independent of the number of generated queries.

We further propose an investigation on the stability of our approach across various training datasets $\mathcal{D}_{\text{Train}}$. Since a training dataset contains a subset of samples of size T drawn from the solution space $\mathcal{S}$, it is possible to build different datasets from the same problem instance by randomizing this samples-drawing procedure.

We present the raw data of each of our metrics obtained using 10 distinct datasets built from the same fixed problem instance. Thus, all the datasets share the same asset universe, cardinality, and seen portion ϵ, and they simply differ for the training bitstrings that get sampled from $P_{\text{Target}}(x)$. Tables 13 - 14 show the results we obtained for the different pre-generalization condition, validity-based and value-based generalization metrics across the 10 different datasets, where each line corresponds to one dataset. We see that the TNBM beats the GAN for all (F, R, C) values. The relative percentage errors across the datasets for (F, R, C) values are smaller for the TNBM $(0.5\%, 0.5\%, 0.5\%)$ than the GAN

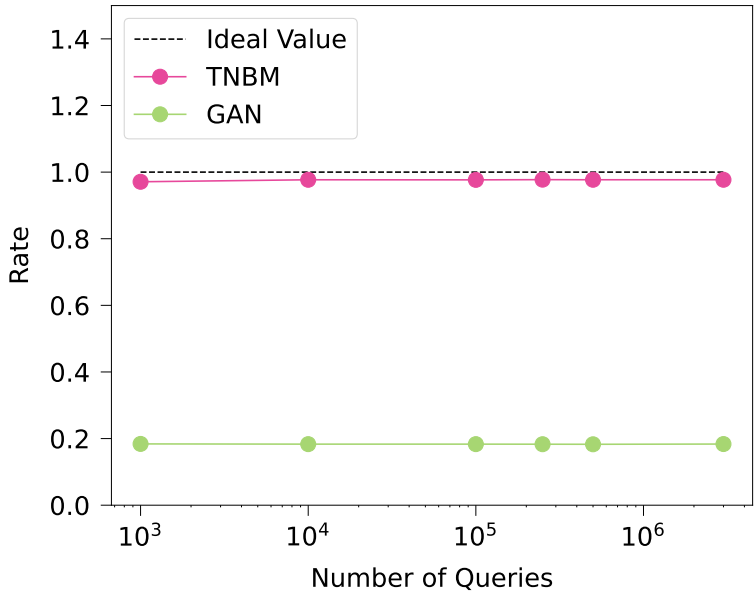


Fig. 36: **Rate trends for increasing number of queries.** The plot displays the constant behaviour of the rate R for both TNBM (pink) and GAN (green) as we increase the number of queries Q retrieved from the trained models. The dashed black line shows the ideal metric value of 1. In both models, the R is independent of the number of generated queries.

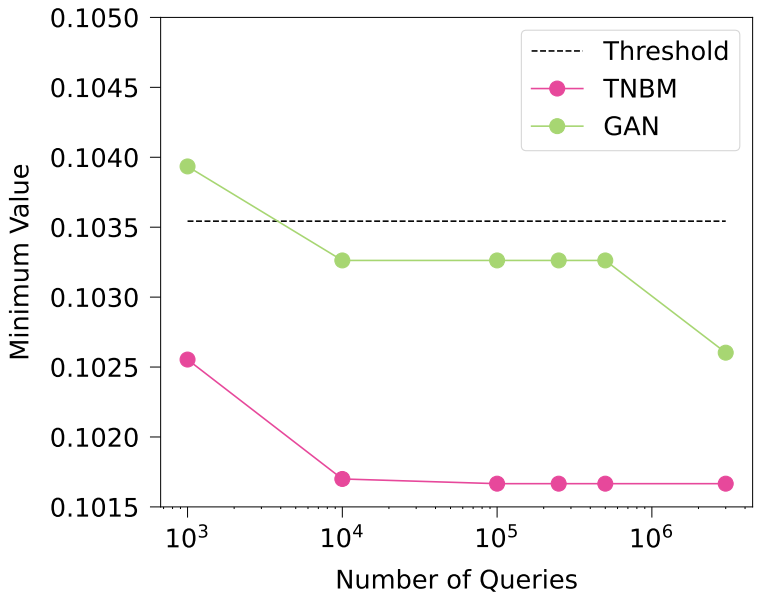


Fig. 37: **Minimum Value (MV) trends for increasing number of queries.** The plot displays the decreasing behaviour of the MV metric for both TNBM (pink) and GAN (green) as we increase the number of queries Q retrieved from the trained models. The dashed black line shows the minimum value of the training set MV. Note that both models not only produce unseen valid samples, but also samples with better quality than those in the training set. As we increase Q, both models are more likely to produce a query with a lower cost value, even if the GAN requires more samples than the TNBM to dip under the threshold.

$(13\%, 13\%, 30\%)$, demonstrating that the TNBM produces more stable results across datasets. However, both standard deviations are small enough to show that our metrics produce similar results across various

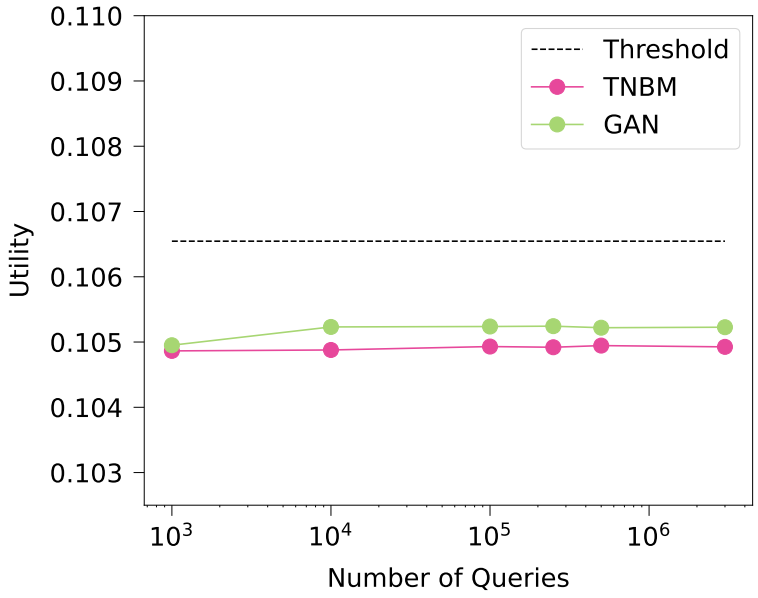


Fig. 38: **Utility trends for increasing number of queries.** The plot displays the constant behaviour of the utility U for both TNBM (pink) and GAN (green) as we increase the number of queries Q retrieved from the trained models. The dashed black line shows the threshold value of the training set U. Both the GAN and TNBM remain under the threshold, independent of the number of queries.

training data.

For the quality-based metrics, we see that the MV for the TNBM is always either equal or less than that of the GAN. However, for U, the TNBM and the GAN trade-off in being the winner. This is not a surprise, as in Table 5 the TNBM and the GAN produced very similar values for the utility. The same argument from Section 5.3.5 holds such that both the TNBM and GAN are able to generate low cost samples. Simply, the TNBM contains more diversified high quality samples, which is not captured by the metric U.

An additional analysis on the stability of the different generative models would be the investigation of their generalization capabilities across different problem instances, especially the ones characterized by larger asset universes, e.g. $N = 500$ (which would correspond to all the assets in the S&P500 index). We highlight here that our approach is not limited to the relatively small universe size considered in this work, i.e. $N = 20$, that was chosen to allow for a practically feasible comparison with quantum generative models in the near term.

10.1.4 Supplementary Figures

We include supplementary figures to further demonstrate some of our results.

E	F	R	C
0.989	0.982	0.971	0.405
0.989	0.978	0.968	0.406
0.989	0.971	0.961	0.401
0.989	0.984	0.973	0.407
0.989	0.983	0.973	0.406
0.989	0.985	0.975	0.407
0.989	0.978	0.967	0.405
0.989	0.977	0.967	0.404
0.989	0.987	0.977	0.406
0.989	0.987	0.977	0.409

Tab. 13: **TNBM pre-generalization and validity-based generalization metrics' values across multiple training datasets from the same problem instance**. We see that the metrics have similar values across the 10 datasets under examination with relative percentage errors $(0.5\%, 0.5\%, 0.5\%)$ for (F, R, C) values respectively. Thus, our metrics produce similar values across multiple training datasets, demonstrating that they are independent of the portion of training samples selected from the valid space.

E	F	R	C
0.999	0.249	0.249	0.0062
0.996	0.236	0.235	0.0062
0.996	0.309	0.307	0.0063
0.998	0.233	0.233	0.0042
0.995	0.181	0.179	0.0049
0.999	0.232	0.232	0.0061
0.997	0.274	0.274	0.0110
0.997	0.276	0.275	0.0071
0.999	0.239	0.239	0.0066
0.994	0.251	0.249	0.0077

Tab. 14: **GAN pre-generalization and validity-based generalization metrics' values across multiple training datasets from the same problem instance**. We see that the metrics have similar values across the 10 datasets under examination with mean standard deviations $(13\%, 13\%, 30\%)$ for (F, R, C) values respectively. Despite not being nearly as stable as the TNBM, we see that our metrics produce similar values across multiple training datasets, demonstrating that they independent of the portion of training samples selected from the valid space.

In Figure 39, we provide a visualization of the GAN quality-based generalization metrics in analogy to Figure 15. By comparing the two plots, we can see that both models reach the low-risk section of the spectrum, but the TNBM samples exhibit more diversity than the GAN ones.

U	U_T	MV	MV_T
0.1049	0.1064	0.1017	0.1018
0.1049	0.1065	0.1017	0.1034
0.1048	0.1067	0.1017	0.1018
0.1049	0.1064	0.1017	0.1031
0.1047	0.1062	0.1017	0.1033
0.1049	0.1065	0.1017	0.1027
0.1051	0.1068	0.1017	0.1036
0.1049	0.1065	0.1017	0.1029
0.1048	0.1064	0.1017	0.1039
0.1049	0.1062	0.1017	0.1021

Tab. 15: **TNBM quality-based metrics' values across various training datasets from the same problem instance.** The second and last columns display the values for the training set, defined as the U and the MV computed for the samples in $\mathcal{D}_{\text{Train}}$. We see that the TNBM's U and MV is always less than the training threshold. Additionally, the same low MV value that exists in the fixed problem universe is generated independent of the training set.

U	U_T	MV	MV_T
0.1041	0.1064	0.1032	0.1018
0.1040	0.1065	0.1021	0.1034
0.1042	0.1067	0.1019	0.1018
0.1038	0.1064	0.1019	0.1031
0.1029	0.1062	0.1017	0.1033
0.1044	0.1065	0.1018	0.1027
0.1043	0.1068	0.1028	0.1036
0.1048	0.1065	0.1024	0.1029
0.1044	0.1064	0.1017	0.1039
0.1056	0.1064	0.1038	0.1021

Tab. 16: **GAN quality-based metrics' values across various training datasets from the same problem instance.** The second and last columns display the values for the training set, defined as the U and the MV computed for the samples in $\mathcal{D}_{\text{Train}}$. We see that the GAN's U is always less than the training threshold; however, this is not always true for MV, as the GAN has a lower MV value only 70% of the time.

In Figure 40 we display a comparison of the training stability of TNBM and all the three GANs considered in this work, showing how good each of the model is in capturing the correct cardinality pattern encoded in the dataset. We can detect the higher instability affecting GAN models, as opposed to the MPS performance, which appears remarkable. We highlight that even if the TNBM produces only queries with a given cardinality, similar to the GAN-MC histograms, the quantum-inspired model is not exhibiting mode collapse onto an unseen and valid bitstring, as the coverage is not negligible as in the GAN-MC case (see Table 4).

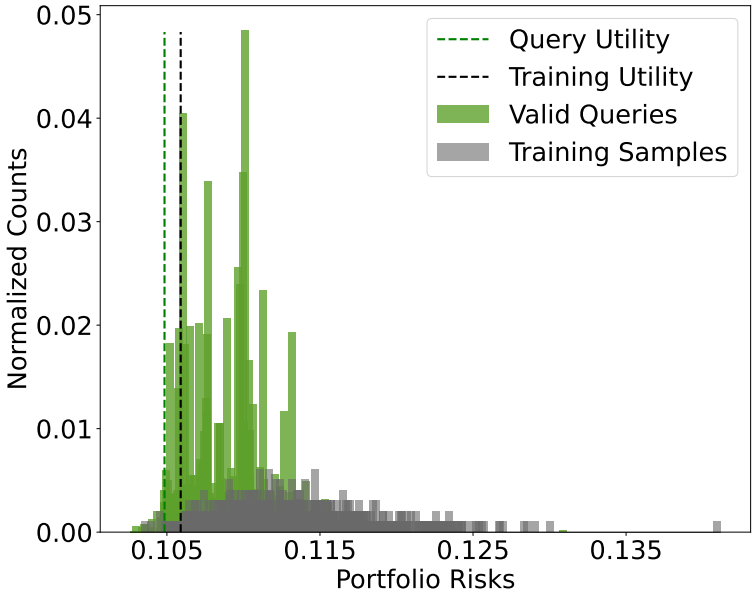


Fig. 39: **Visualization of quality-based metrics for GAN-generated queries.** The plot displays the number of portfolio counts associated to given risk values. The green spikes represent valid GAN queries, whereas the gray spikes represent the samples from the training set. Note that for calculating our metrics, we used $Q = 10^5$ queries, but the training distribution only contains 1,848 samples, hence the need for normalizing the histograms. Here, we set the utility threshold to $t = 5\%$. Similar to the TNBM, the model distribution learns the low-risk 'bias' encoded in the training set, and generates more values of low risk. However, unlike the TNBM, the model frequency counts per query are higher, and the sample diversity is quite low. The queries have a lower utility (green dashed) than the training set (black dashed), thus meeting the condition in Equation 31. Ultimately, no matter the query count, we see that the GAN can reach low risk queries, but simply has less diversity among them in contrast to the TNBM.

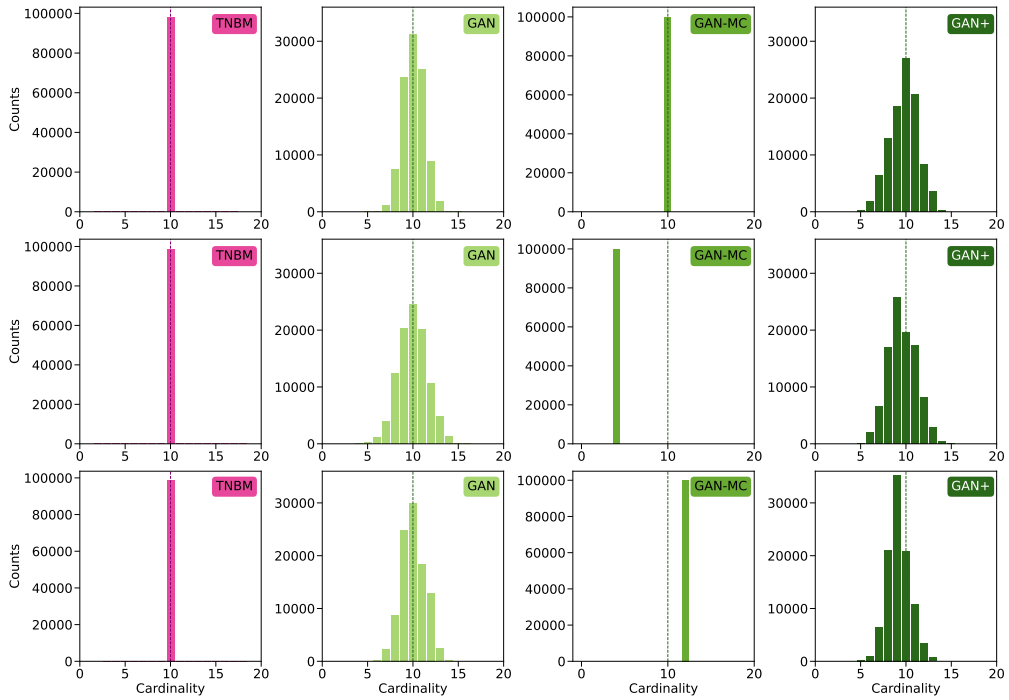


Fig. 40: **Cardinality distributions of queries generated by multiple models during independent trainings.** We represent cardinality histograms obtained when taking $Q = 10^5$ queries from three independently trained instances of each model family (TNBM, GAN, GAN-MC, GAN+). Each plot displays the cardinality distribution of the retrieved queries, along with the desired cardinality $k = 10$. We can see that the three TNBM models generate queries that always learn accurately the cardinality constraint, whereas the GAN models show less training stability, which is known to be one of the issues affecting this class of classical generative models. Specifically, for GAN and GAN+ we see that while each model always produces at least some valid queries, the centers and tails of the distributions vary greatly for each instance. For GAN-MC, distinct trainings collapse onto different cardinalities, implying that the model is not always guaranteed to generate valid queries.

10.2 Appendix Chapter 5

We present supplemental data for the results displayed in 6 for both validity and quality-based generalization performance.

10.2.1 Validity-Based Generalization

Here, we show the metrics' results for the last training iteration in Figure 16 and for the various ϵ values in Figure 17 in Table 17. All metric values are averages over 5 independent trainings, where the error is given by the standard deviation over the square-root of the number of runs. Similarly, the average random baseline values with corresponding errors are shown in Figure 18.

L	ϵ	F	$\tilde{R}$	$\tilde{C}$	p	$\mathrm{KL}_{\mathrm{Target}}$	$\mathrm{KL}_{\mathrm{Train}}$
2	0.1	0.28 ± 0.04	0.29 ± 0.04	0.81 ± 0.04	0.31 ± 0.04	1.6 ± 0.2	3.37 ± 0.07
2	0.3	0.189 ± 0.005	0.248 ± 0.007	0.869 ± 0.007	0.259 ± 0.005	1.46 ± 0.02	2.578 ± 0.006
2	0.5	0.152 ± 0.003	0.263 ± 0.005	0.888 ± 0.007	0.268 ± 0.004	1.418 ± 0.008	2.079 ± 0.008
2	0.7	0.092 ± 0.002	0.251 ± 0.004	0.891 ± 0.009	0.259 ± 0.004	1.427 ± 0.006	1.771 ± 0.006
2	0.9	0.035 ± 0.001	0.27 ± 0.01	0.891 ± 0.03	0.265 ± 0.002	1.403 ± 0.006	1.506 ± 0.005
4	0.1	0.39 ± 0.05	0.41 ± 0.05	0.82 ± 0.03	0.43 ± 0.05	1.4 ± 0.2	2.89 ± 0.07
4	0.3	0.238 ± 0.005	0.300 ± 0.005	0.825 ± 0.009	0.328 ± 0.006	1.47 ± 0.02	2.36 ± 0.01
4	0.5	0.190 ± 0.004	0.309 ± 0.004	0.863 ± 0.007	0.341 ± 0.008	1.33 ± 0.02	1.587 ± 0.008
4	0.7	0.125 ± 0.001	0.316 ± 0.003	0.896 ± 0.009	0.336 ± 0.005	1.266 ± 0.005	1.587 ± 0.008
4	0.9	0.047 ± 0.002	0.32 ± 0.01	0.87 ± 0.01	0.344 ± 0.006	1.237 ± 0.009	1.329 ± 0.009
8	0.1	0.35 ± 0.03	0.35 ± 0.03	0.74 ± 0.03	0.41 ± 0.03	1.6 ± 0.1	2.48 ± 0.02
8	0.3	0.491 ± 0.007	0.546 ± 0.009	0.89 ± 0.01	0.603 ± 0.005	0.86 ± 0.03	1.73 ± 0.01
8	0.5	0.43 ± 0.01	0.57 ± 0.01	0.941 ± 0.004	0.62 ± 0.01	0.69 ± 0.02	1.27 ± 0.02
8	0.7	0.31 ± 0.02	0.57 ± 0.02	0.952 ± 0.004	0.61 ± 0.02	0.66 ± 0.02	0.98 ± 0.02
8	0.9	0.119 ± 0.005	0.55 ± 0.01	0.983 ± 0.005	0.589 ± 0.009	0.662 ± 0.009	0.761 ± 0.009
16	0.1	0.32 ± 0.06	0.30 ± 0.05	0.71 ± 0.02	0.42 ± 0.05	1.2 ± 0.1	2.07 ± 0.02
16	0.3	0.64 ± 0.02	0.67 ± 0.02	0.92 ± 0.01	0.74 ± 0.02	0.61 ± 0.05	1.46 ± 0.02
16	0.5	0.61 ± 0.05	0.69 ± 0.04	0.950 ± 0.009	0.78 ± 0.04	0.46 ± 0.06	1.03 ± 0.04
16	0.7	0.49 ± 0.02	0.708 ± 0.009	0.956 ± 0.005	0.78 ± 0.01	0.43 ± 0.02	0.73 ± 0.02
16	0.9	0.22 ± 0.02	0.65 ± 0.02	0.96 ± 0.01	0.76 ± 0.02	0.45 ± 0.03	0.54 ± 0.03

Tab. 17: **Validity-based generalization values over various circuit depths and training set sizes.** The table lists the average $(F, \tilde{R}, \tilde{C})$ values, along with the precision p and the KL divergences relative to the target and the training set, across 5 independent trainings with the associated error for each model. Note that these are all values computed after sampling from the fully trained model ($i_{\mathrm{Max}} = 10\mathrm{k}$) when learning the cardinality-constrained target distribution.

ϵ	F	$\tilde{R}$	$\tilde{C}$	p
0.1	0.206 ± 0.002	0.224 ± 0.002	0.906 ± 0.003	0.225 ± 0.001
0.3	0.169 ± 0.001	0.225 ± 0.002	0.909 ± 0.003	0.225 ± 0.001
0.5	0.128 ± 0.001	0.228 ± 0.002	0.914 ± 0.003	0.225 ± 0.001
0.7	0.0790 ± 0.0007	0.222 ± 0.002	0.904 ± 0.005	0.225 ± 0.001
0.9	0.0276 ± 0.0005	0.219 ± 0.004	0.904 ± 0.007	0.225 ± 0.001

Tab. 18: **Random search baselines for the validity-based generalization metrics across various ϵ values.** We show the average $(F, \tilde{R}, \tilde{C})$ values alongside the precision p, when no training is conducted and samples are taken at random. The results are averaged from 5 independent runs, and displayed with their corresponding standard errors.

10.2.2 Quality-Based Generalization

We show the metrics' results for each output distribution in Figure 18, containing different degrees of reweighting. Note that in Table 19 all values are averages over 15 independent trainings, where their error is given by the standard deviation over the square-root of the number of runs. In Table 20, we show the median score of $F + \tilde{R} + \tilde{C}$ for each distribution, which directly corresponds to the model outputs displayed in Figure 18.

Distribution	F	$\tilde{R}$	$\tilde{C}$	p
Uniform	0.74 ± 0.07	0.69 ± 0.06	0.76 ± 0.02	0.77 ± 0.06
Reweighted T	0.74 ± 0.07	0.69 ± 0.06	0.74 ± 0.02	0.77 ± 0.06
Reweighted $T/2$	0.85 ± 0.07	0.75 ± 0.05	0.59 ± 0.03	0.87 ± 0.06

Tab. 19: **Average validity-based generalization metrics when training on uniform vs reweighted *Evens* distributions.** We show the average $(F, \tilde{R}, \tilde{C})$ values, along with the precision p across 15 independent runs with the associated error for each model. Note that these are all values computed after sampling from the fully trained model ($i_{\text{Max}} = 10\text{k}$).

Distribution	F	$\tilde{R}$	$\tilde{C}$	p	U	$P_{c<-7}$
Uniform	0.99	0.89	0.81	0.99	-6.88	0.008
Reweighted T	1.0	0.91	0.79	1.0	-6.96	0.008
Reweighted $T/2$	1.0	0.82	0.79	1.0	-8.89	0.056

Tab. 20: **Median validity-based generalization and quality-based metrics when training on uniform vs reweighted *Evens* distributions.** We show the median $(F, \tilde{R}, \tilde{C})$ values, along with the precision p for 15 independent runs. Additionally, we show U and $P_{c<-7}$ values that correspond to the median $(F, \tilde{R}, \tilde{C})$ score. Note that these are all values computed after sampling from the fully trained model ($i_{\text{Max}} = 10\text{k}$).

10.3 Appendix Chapter 6

10.3.1 Training Details

Here, we show the training loss curve that provides the optimal weights and biases to utilize for our model on the H1 hardware and emulator devices.

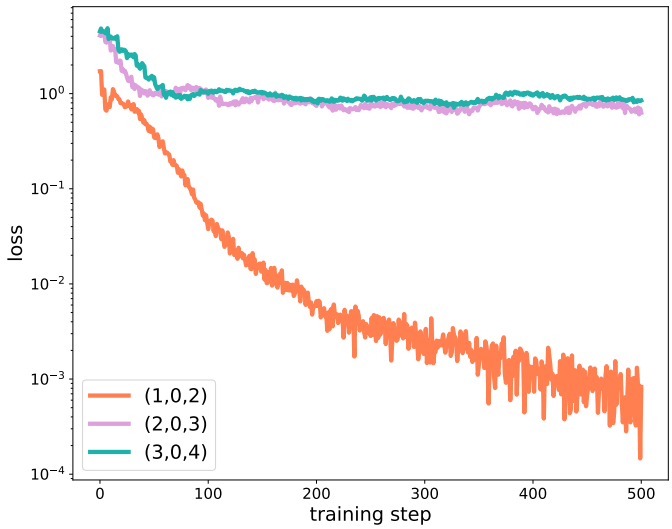


Fig. 41: **The KL training curves for all model sizes on the statevector simulator.** Here, we show the convergence of a $(1, 0, 2)$ (orange), a $(2, 0, 3)$ (pink), and a $(3, 0, 4)$ (green) QNBM algorithm. The parameters utilized for the hardware circuit are taken from the last iteration step. Consequently, the output probability distribution at the final training step becomes the target distribution for stress testing the H1 hardware/emulator.

References

[1] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 2019.

[2] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 2014.

[3] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 2020.

[4] Bela Bauer, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. Quantum algorithms for quantum chemistry and quantum materials science. *Chemical Reviews*, 2020.

[5] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. Quantum chemistry in the age of quantum computing. *Chemical Reviews*, 2019.

[6] Seunghoon Lee, Joonho Lee, Huanchen Zhai, Yu Tong, Alexander M. Dalzell, Ashutosh Kumar, Phillip Helms, Johnnie Gray, Zhi-Hao Cui, Wenyuan Liu, Michael Kastoryano, Ryan Babbush, John Preskill, David R. Reichman, Earl T. Campbell, Edward F. Valeev, Lin Lin, and Garnet Kin-Lic Chan. Is there evidence for exponential quantum advantage in quantum chemistry? *arXiv: 2208.02199*, 2022.

[7] Ivan Kassal, Stephen P Jordan, Peter J Love, Masoud Mohseni, and Alán Aspuru-Guzik. Polynomial-time quantum algorithm for the simulation of chemical dynamics. *Proc. Natl. Acad. Sci.*, 2008.

[8] Andrew J Daley, Immanuel Bloch, Christian Kokail, Stuart Flannigan, Natalie Pearson, Matthias Troyer, and Peter Zoller. Practical quantum advantage in quantum simulation. *Nature*, 2022.

[9] Fergus Barratt, James Dborin, Matthias Bal, Vid Stojevic, Frank Pollmann, and Andrew G Green. Parallel quantum simulation of

large systems on small nisq computers. *npj Quantum Information*, 2021.

[10] Alejandro Perdomo-Ortiz, Marcello Benedetti, John Realpe-Gómez, and Rupak Biswas. Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers. *Quantum Science and Technology*, 2018.

[11] Maria Schuld and Nathan Killoran. Is quantum advantage the right goal for quantum machine learning? *PRX Quantum*, 2022.

[12] Marco Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2022.

[13] Junde Li, Rasit Topaloglu, and Swaroop Ghosh. Quantum generative models for small molecule drug discovery. *arXiv: 2101.03438*, 2021.

[14] Nathan Brown, Marco Fiscato, Marwin H.S. Segler, and Alain C. Vaucher. Guacamol: Benchmarking models for de novo molecular design. *Journal of Chemical Information and Modeling*, 2019.

[15] He Huang, Philip S. Yu, and Changhu Wang. An introduction to image synthesis with generative adversarial nets. *arXiv: 1803.04469*, 2018.

[16] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[17] Hao Peng, Christopher S. Gates, Bhaskar Pratim Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[18] Ahmed M Alaa, Boris van Breugel, Evgeny Saveliev, and Mihaela van der Schaar. How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models. *arXiv: 2102.08921*, 2021.

[19] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised generative modeling using matrix product states. *PRX*, 2018.

[20] Marcello Benedetti, Delfina Garcia-Pintos, Yunseong Nam, and Alejandro Perdomo-Ortiz. A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Information*, 2018.

[21] Brian Coyle, Daniel Mills, Vincent Danos, and Elham Kashefi. The born supremacy: quantum advantage and training of an ising born machine. *npj Quantum Information*, 2020.

[22] Sachin Kasture, Oleksandr Kyriienko, and Vincent E Elfving. Protocols for classically training quantum generative models on probability distributions. *arXiv: 2210.13442*, 2022.

[23] Walter Winci, Lorenzo Buffoni, Hossein Sadeghi, Amir Khoshaman, Evgeny Andriyash, and Mohammad H Amin. A path towards quantum advantage in training deep generative models with quantum annealers. *Machine Learning: Science and Technology*, 2020.

[24] Frank Phillipson. Quantum machine learning: Benefits and practical examples. In *QANSWER*, pages 51–56, 2020.

[25] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 2018.

[26] Jin-Guo Liu and Lei Wang. Differentiable learning of quantum circuit born machines. *Physical Review A*, 2018.

[27] Xun Gao, Zhengyu Zhang, and Luming Duan. A quantum machine learning algorithm based on generative models. *Science Advances*, 2018.

[28] Manuel S. Rudolph, Ntwali Bashige Toussaint, Amara Katabarwa, Sonika Johri, Borja Peropadre, and Alejandro Perdomo-Ortiz. Generation of high-resolution handwritten digits with an ion-trap quantum computer. *Phys. Rev. X*, 2022.

[29] Marcel Hinsche, Marios Ioannou, Alexander Nietner, Jonas Haferkamp, Yihui Quek, Dominik Hangleiter, Jean-Pierre Seifert, Jens Eisert, and Ryan Sweke. Learnability of the output distributions of local quantum circuits. *arXiv: 2110.05517*, 2021.

[30] Ryan Sweke, Jean-Pierre Seifert, Dominik Hangleiter, and Jens Eisert. On the quantum versus classical learnability of discrete distributions. *Quantum*, 2021.

[31] Marcel Hinsche, Marios Ioannou, Alexander Nietner, Jonas Haferkamp, Yihui Quek, Dominik Hangleiter, Jean-Pierre Seifert, Jens Eisert, and Ryan Sweke. A single t-gate makes distribution learning hard. *arXiv: 2207.03140*, 2022.

[32] Kaitlin Gili, Marta Mauri, and Alejandro Perdomo-Ortiz. Generalization metrics for practical quantum advantage in generative models. *arXiv: 2201.08770*, 2023.

[33] Kaitlin Gili, Mohamed Hibat-Allah, Marta Mauri, Chris Ballance, and Alejandro Perdomo-Ortiz. Do quantum circuit born machines generalize? *Quantum Sci. Technol.*, 2023.

[34] Kaitlin Gili, Mykolas Sveistrys, and Chris Ballance. Introducing nonlinear activations into quantum generative models. *Phys. Rev. A*, 2023.

[35] Kaitlin Gili, Rohan S. Kumar, Mykolas Sveistrys, and C. J. Ballance. Generative modeling with quantum neurons. *arXiv: 2302.00788*, 2023.

[36] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 2014.

[38] Andrei Cristian Nica, Moksh Jain, Emmanuel Bengio, Cheng-Hao Liu, Maksym Korablyov, Michael M. Bronstein, and Yoshua Bengio. Evaluating generalization in GFlownets for molecule design. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

[39] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, 2020.

[40] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. A review on generative adversarial networks: Algorithms, theory, and applications. *arXiv: 2001.06937*, 2020.

[41] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling. *GAMM-Mitteilungen*, 2021.

[42] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.

[43] Francois-Xavier Briol, Alessandro Barp, Andrew B. Duncan, and Mark Girolami. Statistical inference for generative models with maximum mean discrepancy. *arXiv: 1906.05944*, 2019.

[44] Yuxuan Du, Zhuozhuo Tu, Bujiao Wu, Xiao Yuan, and Dacheng Tao. Power of quantum generative learning. *arXiv: 2205.04730*, 2022.

[45] Shengjia Zhao, Hongyu Ren, Arianna Yuan, Jiaming Song, Noah Goodman, and Stefano Ermon. Bias and generalization in deep generative models: An empirical study. *Advances in Neural Information Processing Systems*, 31, 2018.

[46] Ali Borji. Pros and cons of gan evaluation measures: New developments. *Computer Vision and Image Understanding*, 2022.

[47] Hoang Thanh-Tung and Truyen Tran. Toward a generalization metric for deep generative models. *arXiv: 2011.00754*, 2021.

[48] Claude Sammut and Geoffrey I. Webb, editors. *Probably Approximately Correct Learning*, pages 805–805. Springer US, Boston, MA, 2010.

[49] Javier Alcazar, Mohammad Ghazi Vakili, Can B. Kalayci, and Alejandro Perdomo-Ortiz. Geo: Enhancing combinatorial optimization with classical and quantum generative models. *arXiv: 2101.06250*, 2021.

[50] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *Advances in Neural Information Processing Systems*, 2016.

[51] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv: 1706.08500*, 2018.

[52] Mikołaj Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. *arXiv: 1801.01401*, 2021.

[53] Ishaan Gulrajani, Colin Raffel, and Luke Metz. Towards GAN benchmarks which require generalization. *arXiv: 2001.03653*, 2020.

[54] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. In *NeurIPS*, 2018.

[55] Loïc Simon, Ryan Webster, and Julien Rabin. Revisiting precision and recall definition for generative model evaluation. *arXiv: 1905.05441*, 2019.

[56] Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. Reliable fidelity and diversity metrics for generative models. In *International Conference on Machine Learning*, pages 7176–7185. PMLR, 2020.

[57] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *arXiv: 1904.06991*, 2019.

[58] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv: 1511.01844*, 2016.

[59] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *arXiv: 1806.07755*, 2018.

[60] Ryan Webster, Julien Rabin, Loïc Simon, and Frédéric Jurie. Detecting overfitting of deep generative networks via latent recovery. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11265–11274, 2019.

[61] Casey Meehan, Kamalika Chaudhuri, and Sanjoy Dasgupta. A non-parametric test to detect data-copying in generative models. *arXiv: 2004.05675*, 2020.

[62] Tai-Danae Bradley, E Miles Stoudenmire, and John Terilla. Modeling sequences with quantum states: a look under the hood. *Machine Learning: Science and Technology*, 2020.

[63] James Stokes and John Terilla. Probabilistic modeling with matrix product states. *Entropy*, 2019.

[64] Jacob Miller, Guillaume Rabusseau, and John Terilla. Tensor networks for probabilistic sequence modeling. *arXiv: 2003.01039*, 2020.

[65] Jan Benhelm, Gerhard Kirchmair, Christian F. Roos, and Rainer Blatt. Towards fault-tolerant quantum computing with trapped ions. *Nature Physics*, 2008.

[66] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 2020.

[67] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 2021.

[68] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2018.

[69] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum algorithms. *Rev. Mod. Phys.*, 2022.

[70] Javier Alcazar, Vicente Leyton-Ortega, and Alejandro Perdomo-Ortiz. Classical versus quantum models in machine learning: insights from a finance application. *Machine Learning: Science and Technology*, 2020.

[71] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 2019.

[72] Manuel S. Rudolph, Jacob Miller, Jing Chen, Atithi Acharya, and Alejandro Perdomo-Ortiz. Synergy between quantum circuits and tensor networks: Short-cutting the race to practical quantum advantage. *arXiv: 2208.13673*, 2022.

[73] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. Expressive power of parametrized quantum circuits. *Physical Review Research*, 2020.

[74] Daiwei Zhu, Norbert M Linke, Marcello Benedetti, Kevin A Landsman, Nhung H Nguyen, C Huerta Alderete, Alejandro Perdomo-Ortiz, Nathan Korda, A Garfoot, Charles Brecque, et al.

Training of quantum circuits on a hybrid quantum computer. *Science Advances*, 2019.

[75] Daiwei Zhu, Weiwei Shen, Annarita Giani, Saikat Ray Majumder, Bogdan Neculaes, and Sonika Johri. Copula-based risk aggregation with trapped ion quantum computers. *arXiv: 2206.11937*, 2022.

[76] Ali Rad, Alireza Seif, and Norbert M. Linke. Surviving the barren plateau in variational quantum circuits with bayesian learning initialization. *arXiv: 2203.02464*, 2022.

[77] James Joyce. *Kullback-Leibler Divergence*, pages 720–722. 2011.

[78] Chiara Leadbeater, Louis Sharrock, Brian Coyle, and Marcello Benedetti. F-divergences and cost function locality in generative modelling with quantum circuits. *Entropy*, 2021.

[79] Manuel S Rudolph, Jing Chen, Jacob Miller, Atithi Acharya, and Alejandro Perdomo-Ortiz. Decomposition of matrix product states into shallow quantum circuits. *arXiv: 2209.00595*, 2022.

[80] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 2021.

[81] Lasse Bjørn Kristensen, Matthias Degroote, Peter Wittek, Alán Aspuru-Guzik, and Nikolaj T. Zinner. An artificial spiking quantum neuron. *arXiv: 1907.06269*, 2020.

[82] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. *arXiv: 1711.11240*, 2017.

[83] Adam Paetznick and Krysta M. Svore. Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries. *arXiv: 1311.1074*, 2013.

[84] Alex Bocharov, Martin Roetteler, and Krysta M. Svore. Efficient synthesis of universal repeat-until-success quantum circuits. *Physical Review Letters*, 2015.

[85] Yulin Wu, Wan-Su Bao, Sirui Cao, Fusheng Chen, Ming-Cheng Chen, Xiawei Chen, Tung-Hsun Chung, Hui Deng, Yajie Du, Daojin Fan, et al. Strong quantum computational advantage using a superconducting quantum processor. *Physical Review Letters*, 2021.

[86] Lars S Madsen, Fabian Laudenbach, Mohsen Falamarzi Askarani, Fabien Rortais, Trevor Vincent, Jacob FF Bulmer, Filippo M Miatto, Leonhard Neuhaus, Lukas G Helt, Matthew J Collins, et al. Quantum computational advantage with a programmable photonic processor. *Nature*, 2022.

[87] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *Nature Physics*, 2018.

[88] Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. On the complexity and verification of quantum random circuit sampling. *Nature Physics*, 2019.

[89] Marcela Carena, Henry Lamm, Ying-Ying Li, Joseph D Lykken, Lian-Tao Wang, and Yukari Yamauchi. Practical quantum advantages in high energy physics. SnowMass, 2021.

[90] Paul Alsing, Phil Battle, Joshua C Bienfang, Tammie Borders, Tina Brower-Thomas, Lincoln Carr, Fred Chong, Siamak Dadras, Brian DeMarco, Ivan Deutsch, et al. Accelerating progress towards practical quantum advantage: A national science foundation project scoping workshop. *arXiv: 2210.14757*, 2022.

[91] Troels F. Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V. Isakov, David Wecker, John M. Martinis, Daniel A. Lidar, and Matthias Troyer. Defining and detecting quantum speedup. *Science*, 2014.

[92] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *arXiv: 2106.04399*, 2021.

[93] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[94] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 1952.

[95] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in Neural Information Processing Systems*, pages 700–709, 2018.

[96] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[97] Tong Che, Yanran Li, Athul Jacob, Y. Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv: 1612.02136*, 2016.

[98] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223, 2017.

[99] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. In *NIPS*, 2017.

[100] Expected coverage after sampling with replacement k times. *Mathematics Stack Exchange*, 2017.

[101] Probability distribution of coverage of a set after x independently, randomly selected members of the set. *Mathematics Stack Exchange*, May 2018.

[102] Rubén Ruiz-Torrubiano, Sergio García-Moratilla, and Alberto Suárez. Optimization problems with cardinality constraints. In *Computational Intelligence in Optimization: Applications and Implementations*, pages 105–130. Springer, 2010.

[103] C. Moussa, H. Wang, M. Araya-Polo, T. Bäck, and V. Dunjko. Application of quantum-inspired generative models to small molecular datasets. *arXiv: 2304.10867*, 2023.

[104] Mohamed Hibat-Allah, Marta Mauri, Juan Carrasquilla, and Alejandro Perdomo-Ortiz. A framework for demonstrating practical quantum advantage: Racing quantum against classical generative models. *arXiv: 2303.15626*, 2023.

[105] Kathleen E Hamilton, Eugene F Dumitrescu, and Raphael C Pooser. Generative model benchmarks for superconducting qubits. *Physical Review A*, 2019.

[106] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 2019.

[107] Brian Coyle, Maxwell Henderson, Justin Chan Jin Le, Niraj Kumar, Marco Paini, and Elham Kashefi. Quantum versus classical generative modelling in finance. *Quantum Science and Technology*, 2021.

[108] Marcello Benedetti, Brian Coyle, Mattia Fiorentini, Michael Lubasch, and Matthias Rosenkranz. Variational inference with a quantum computer. *Phys. Rev. Applied*, 2021.

[109] Ivan Glasser, Ryan Sweke, Nicola Pancotti, Jens Eisert, and Ignacio Cirac. Expressive power of tensor-network factorizations for probabilistic modeling. *Advances in Neural Information Processing Systems*, 2019.

[110] Xun Gao, Eric R. Anschuetz, Sheng-Tao Wang, J. Ignacio Cirac, and Mikhail D. Lukin. Enhancing generative models via quantum correlations. *Phys. Rev. X*, 2022.

[111] Yuxuan Du, Zhuozhuo Tu, Bujiao Wu, Xiao Yuan, and Dacheng Tao. Theory of quantum generative learning models with maximum mean discrepancy. *arXiv: 2205.04730*, 2022.

[112] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv: 1505.03906*, 2015.

[113] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., NY, 1979.

[114] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv: 1604.00772*, 2016.

[115] Yasunari Suzuki, Yoshiaki Kawase, Yuya Masumura, Yuria Hiraga, Masahiro Nakadai, Jiabao Chen, Ken M. Nakanishi, Kosuke Mitarai, Ryosuke Imai, Shiro Tamiya, Takahiro Yamamoto, Tennin Yan, Toru Kawakubo, Yuya O. Nakagawa, Yohei Ibe, Youyuan Zhang, Hirotsugu Yamashita, Hikaru Yoshimura, Akihiro Hayashi, and Keisuke Fujii. Qulacs: a fast and versatile quantum circuit simulator for research purpose. *Quantum*, 2021.

[116] Jarrod Mcclean, Sergio Boixo, Vadim Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 2018.

[117] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature communications*, 2021.

[118] Zoë Holmes, Kunal Sharma, Marco Cerezo, and Patrick J Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum*, 2022.

[119] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021.

[120] Jinkai Tian, Xiaoyu Sun, Yuxuan Du, Shanshan Zhao, Qing Liu, Kaining Zhang, Wei Yi, Wanrong Huang, Chaoyue Wang, Xingyao Wu, Min-Hsiu Hsieh, Tongliang Liu, Wenjing Yang, and Dacheng Tao. Recent advances for quantum neural networks in generative learning. *arXiv: 2206.03066*, 2022.

[121] Matthew DeCross, Eli Chertkov, Megan Kohagen, and Michael Foss-Feig. Qubit-reuse compilation with mid-circuit measurement and reset. *arXiv: 2210.08039*, 2022.

[122] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, and et al. Demonstration of the trapped-ion quantum ccd computer architecture. *Nature*, 2021.

[123] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. PNAS, 1999.

[124] Geoffrey E. Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[125] Nathan Wiebe and Martin Roetteler. Quantum arithmetic and numerical analysis using repeat-until-success circuits. *arXiv: 1406.2040*, 2014.

[126] S. A. Moses, C. H. Baldwin, M. S. Allman, R. Ancona, L. Ascarrunz, C. Barnes, J. Bartolotta, B. Bjork, P. Blanchard, and et al. M. Bohn. A race track trapped-ion quantum processor. *arXiv: 2305.03828*, 2023.

[127] Charles H. Baldwin, Karl Mayer, Natalie C. Brown, Ciarán Ryan-Anderson, and David Hayes. Re-examining the quantum volume test:

Ideal distributions, compiler optimizations, confidence intervals, and scalable resource estimations. *Quantum*, 2022.

[128] Quantinuum H1-1. 2023. https://www.quantinuum.com/.

[129] Alexander J. McCaskey, Zachary P. Parks, Jacek Jakowski, Shirley V. Moore, Titus D. Morris, Travis S. Humble, and Raphael C. Pooser. Quantum chemistry as a benchmark for near-term quantum computers. *npj Quantum Information*, 2019.

[130] Junchao Wang, Guoping Guo, and Zheng Shan. Sok: Benchmarking the performance of a quantum computer. *Entropy*, 2022.

[131] Mohammad Kordzanganeh, Markus Buchberger, Maxim Povolotskii, Wilhelm Fischer, Andrii Kurkin, Wilfrid Somogyi, Asel Sagingalieva, Markus Pflitsch, and Alexey Melnikov. Benchmarking simulated and physical quantum processing units using quantum and hybrid algorithms. *arXiv: 2211.15631*, 2022.

[132] Julian Obst, Johanna Barzen, Martin Beisel, Frank Leymann, Marie Salm, and Felix Truger. Comparing quantum service offerings: A case study of qaoa for maxcut. *arXiv: 2304.12718*, 2023.

[133] Koen Mesman, Zaid Al-Ars, and Matthias Möller. Qpack: Quantum approximate optimization algorithms as universal benchmark for quantum computers. *arXiv: 2103.17193*, 2022.

[134] Khaled M. Mustafa, Rafa E. Al-Qutaish, and Mohammad I. Muhairat. Classification of software testing tools based on the software testing methods. In *2009 Second International Conference on Computer and Electrical Engineering*, pages 229–233, 2009.

[135] Manuel S. Rudolph, Sukin Sim, Asad Raza, Michal Stechly, Jarrod R. McClean, Eric R. Anschuetz, Luis Serrano, and Alejandro Perdomo-Ortiz. Orqviz: Visualizing high-dimensional landscapes in variational quantum algorithms. *arXiv: 2111.04695*, 2021.

[136] Jie Cheng, David A. Bell, and Weiru Liu. An algorithm for bayesian network construction from data. In David Madigan and Padhraic Smyth, editors, *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, pages 83–90, 1997.

[137] Peter D Bruza, Zheng Wang, and Jerome R Busemeyer. Quantum cognition: a new theoretical approach to psychology. *Trends in cognitive sciences*, 19(7):383–393, 2015.

[138] Emmanuel M Pothos and Jerome R Busemeyer. Quantum cognition. *Annual review of psychology*, 73:749–778, 2022.

[139] Samson Wang, Enrico Fontana, Marco Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J Coles. Noise-induced barren plateaus in variational quantum algorithms. *Nature communications*, 2021.

[140] Andrew Arrasmith, Zoë Holmes, Marco Cerezo, and Patrick J Coles. Equivalence of quantum barren plateaus to cost concentration and narrow gorges. *arXiv: 2104.05868*, 2021.

[141] Amira Abbas, Robbie King, Hsin-Yuan Huang, William J. Huggins, Ramis Movassagh, Dar Gilboa, and Jarrod R. McClean. On quantum backpropagation, information reuse, and cheating measurement collapse. *arXiv: 2305.13362*, 2023.

[142] Manuel S. Rudolph, Sacha Lerch, Supanut Thanasilp, Oriel Kiss, Sofia Vallecorsa, Michele Grossi, and Zoë Holmes. Trainability barriers and opportunities in quantum generative modeling. *arXiv: 2305.02881*, 2023.

[143] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, 2021.

[144] Seokwon Yoo, Jeongho Bang, Changhyoup Lee, and Jinhyoung Lee. A quantum speedup in machine learning: finding anin/i-bit boolean function for a classification. *New Journal of Physics*, 2014.

[145] Franz J. Schreiber, Jens Eisert, and Johannes Jakob Meyer. Classical surrogates for quantum learning models. *arXiv: 2206.11740*, 2022.

[146] Haoyuan Cai, Qi Ye, and Dong-Ling Deng. Sample complexity of learning parametric quantum circuits. *arXiv: 2107.09078*, 2022.

[147] Joseph Bowles, Victoria J Wright, Máté Farkas, Nathan Killoran, and Maria Schuld. Contextuality and inductive bias in quantum machine learning. *arXiv: 2302.01365*, 2023.

[148] Jonas M. Kübler, Simon Buchholz, and Bernhard Schölkopf. The inductive bias of quantum kernels. 2021.

[149] Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. *arXiv: 2011.15091*, 2022.

[150] David Haussler. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial Intelligence*, 1988.

[151] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv: 1806.01261*, 2018.

[152] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv: 1511.08458*, 2015.

[153] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv: 1412.6980*, 2014.