Hybrid discrete-continuous compilation of trapped-ion quantum circuits with deep reinforcement learning

Francesco Preti[1,2], Michael Schilling[1,2], Sofiene Jerbi[3,4], Lea M. Trenkwalder[3], Hendrik Poulsen Nautrup[3], Felix Motzoi[1,2], and Hans J. Briegel[3]

[1]Forschungszentrum Jülich, Institute of Quantum Control (PGI-8), D-52425 Jülich, Germany
[2]University of Cologne, Institute of Theoretical Physics, , D-50937 Köln, Germany
[3]University of Innsbruck, Institute for Theoretical Physics, A-6020 Innsbruck, Austria
[4]Freie Universität Berlin, Dahlem Center for Complex Quantum Systems, D-14195 Berlin, Germany.
03-05-2024

Shortening quantum circuits is crucial to reducing the destructive effect of environmental decoherence and enabling useful algorithms. Here, we demonstrate an improvement in such compilation tasks via a combination of using hybrid discrete-continuous optimization across a continuous gate set, and architecture-tailored implementation. The continuous parameters are discovered with a gradient-based optimization algorithm, while in tandem the optimal gate orderings are learned via a deep reinforcement learning algorithm, based on projective simulation. To test this approach, we introduce a framework to simulate collective gates in trapped-ion systems efficiently on a classical device. The algorithm proves able to significantly reduce the size of relevant quantum circuits for trapped-ion computing. Furthermore, we show that our framework can also be applied to an experimental setup whose goal is to reproduce an unknown unitary process.

1 Introduction

The last decade has seen significant progress in the development of quantum computing architectures [1]. While scalable fault-tolerant quantum computers are still out of reach in the near future, noisy, intermediate-scale quantum (NISQ) computers may already offer some benefits over classical ones for specific computational tasks [2, 3]. In particular, variational algorithms [4, 5], where most of the operations depend on several continuous parameters, have emerged as a suitable class of methods that could potentially achieve quantum speed-up on NISQ devices.

Implementing a high-level quantum algorithm both on fault-tolerant and NISQ devices requires adequate methods to compile it in the set of universal quantum gates available to the hardware. While several frameworks for compilation of quantum circuit-based algorithms on physical platforms are being developed [6, 7, 8], common available approaches, such as heuristic and automated search [9, 10], in many case cannot output an optimal circuit for a specific target operation [11].

In digital quantum computers, compilers implement a general quantum circuit through a discrete set of universal quantum gates. In real physical quantum devices, however, and more generally in analog computation, an additional layer of complexity is present, due to the necessity of optimizing continuous gate parameters to reproduce target unitaries. These parameters may depend, e.g., on the specific Hamiltonian employed in the quantum computing platform, such as the XY-Hamiltonian or the Mølmer–Sørensen interaction for trapped ions [12, 13], the cross resonance interaction for IBM quantum computers [14] or the Fermi-Hubbard model for neutral atoms [15]. As a result, when taking into account physical parameters, variational algorithms, and generally continuous gate sets [16], one must supplement the circuit compilation task with a subsequent optimization of the parameters defining the individual constituent gates. For the optimization of continuous parameters, we have several options available, e.g., gradient-based algorithms [17], evolutionary algorithms [18] and direct search [19]. For the compilation of the circuit gate structure, a standard approach is given by methods based on the Solovay-Kitaev algorithm [20], different circuit factorization strategies [21, 22], graph path traversal algorithms, such as the A^* algorithm [23], semi-definite programming and and various machine learning methods, including reinforcement learning [11]. More specifically, deep reinforcement learning has been recently successfully implemented for the optimization of discrete quantum circuits [24, 25, 26, 27, 28, 29, 30].

In reinforcement learning (RL) [31], an agent learns to maximize a properly engineered reward signal by interacting with an environment, which encodes the optimization task to solve. RL has already been applied to solve various challenging tasks, including, e.g., surpassing human performance in certain classes

of games [32] or in complex, computationally expensive problems such as protein folding [33]. Projective Simulation (PS) is a physics-inspired framework for intelligent agents which has also been applied to solve RL tasks in quantum physics [34, 35, 36, 37, 38, 39] and biology [40]. This model can naturally be extended to a deep RL model [41, 42] and has found applications in representation learning [43, 44].

In this work, we propose a unified approach to optimizing the placement and parameter optimization of the gates in the circuit. We argue that such an approach is both relevant to traditional compilation of a more expressive, continuous gateset [16], as well as to variational circuits where the experimental cost-function optimization may be simultaneously performed over discrete and continuous degrees of freedom. We use a RL agent, based on the PS framework, for the combinatorial optimization and a gradient-based optimizer for the continuous optimization of the gate angles. In particular, we extend the method proposed in [26] for the optimization of variational circuits in quantum chemistry and [27] for state preparation to the case of unitary compilation. We consider the framework where an agent, that has control over an experimental platform, interacts with a black-box unitary process and attempts to simulate it. The task of the RL agent is to optimize the position of the gates on the circuit, whereas the gradient-based optimizer finds the optimal set of continuous parameters that minimize a given cost function. The reward function for the RL agent is constructed based on the results of the continuous optimization.

We test the learning algorithm first on standard unitaries, such as Toffoli gates, and then consider the task of quantum process simulation. The latter can be conceived as an experimental black-box unitary approximation strategy that allows the agent to reconstruct an unknown unitary process by compiling a proper quantum circuit.

Independently of our circuit compilation results, we propose a method to speed-up the simulation of quantum circuits based on an efficient representation of trapped-ion gates replacing standard matrix exponentiation. This method allows us to obtain analytic expressions for the ion-trap gates and their gradients and also to simulate the given gate set on other quantum computing platforms, such as superconducting quantum circuits [45] or neutral atoms [46].

The paper is organized as follows: In Section 2 we introduce the quantum circuit framework for trapped-ions. In Section 3.1 we present our method to compute fast analytic ion gates in simulation. In Section 3.2 we discuss continuous optimization methods and strategies to compute the gradients of the cost function both in simulation and on a real quantum device. In Section 3.3 we introduce PS and its extensions in the context of RL methods. In Section 4 we introduce the problem of circuit synthesis and our hybrid RL-continuous optimization method. In Section 5 we discuss the results of applying the proposed method to the compilation of (black-box) unitaries.

2 Problem setting

In this work, we consider a specific set of gates, normally implemented in trapped-ion quantum circuits, which is based on global Mølmer–Sørensen (MS) gates, equatorial rotations acting on the entire register of qubits, as well as local polar rotations. Trapped ions are among the most promising platforms for quantum computing hardware [47]. They exhibit impressive coherence times even in absence of dynamical decoupling and spin echo techniques [48] and have been shown to allow for high-fidelity quantum gates [49, 50]. In a trapped-ion quantum computer, ions – usually $^{43}Ca^{+}$ – are confined in a Paul trap [51] using a varying electromagnetic field. The ions are addressed individually by a system of lasers aligned externally to the trap. The interaction of the laser field with the ion motional and electronic degrees of freedom allows for entangling operations. The laser pulses can be engineered to define the following universal set of quantum gates [52, 53, 54]:

$$\mathrm{MS}(\theta,\phi) = \exp\left\{-i\frac{\theta}{4}(S_x\cos\phi + S_y\sin\phi)^2\right\} \tag{1}$$

$$C_{xy}(\theta,\phi) = \exp\left\{-i\frac{\theta}{2}(S_x\cos\phi + S_y\sin\phi)\right\} \tag{2}$$

$$Z_j(\theta) = \exp\left\{-i\frac{\theta}{2}\sigma_z^{(j)}\right\}, \tag{3}$$

where the first gate is a global MS gate [12, 13], the second gate is a rotation of the entire register of qubits in the equatorial plane of the Bloch sphere, and the third gate represents a single-qubit, and therefore local, σ_z rotation acting on qubit j. The operators $S_x = \sum_{i=1}^{n}\sigma_x^{(i)}$, $S_y = \sum_{i=1}^{n}\sigma_y^{(i)}$, $S_z = \sum_{i=1}^{n}\sigma_z^{(i)}$ are given by the Pauli operators acting on qubit i. These gates can be easily generalized to the qudit case [55]. For the optimization of unitaries, the gate overlap fidelity is a standard figure of merit [56]:

$$F(U,V) = \frac{1}{d^2}|\operatorname{tr}\{VU^\dagger\}|^2, \tag{4}$$

where U and V are unitaries (one of them is the goal of the optimization) and d is the dimension of the Hilbert space – for n-qubit systems $d = 2^n$. Commonly, synthesising quantum circuits to reproduce an arbitrary unitary U requires having access to quantum computing hardware that implements a universal gate set and running the optimization of the continuous parameters.

If the circuit synthesis is performed on a classical computer, it is also necessary to simulate the gate set efficiently. Simulating and optimizing n-qubit collective gates such as those given in Eq. (1) and Eq. (2) is

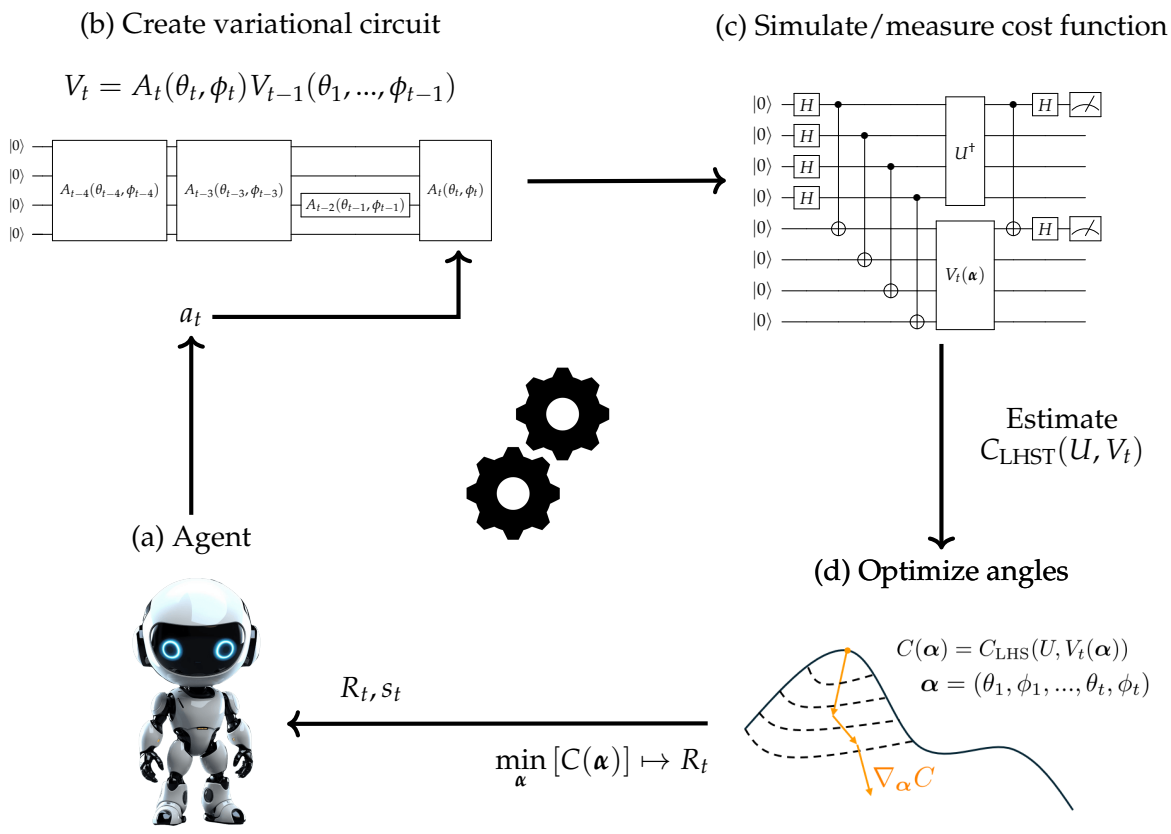


Figure 1: General scheme of the proposed hybrid training loop, where the RL agent (a) learns to optimize a (variational) quantum circuit. By choosing an action $a_t \in \{1, 2, ..., n+2\}$ the gate G_{a_t} is placed on the circuit (b), where $G_1 = \text{MS}, G_2 = C_{xy}, G_3 = Z_1, ..., G_{n+2} = Z_n$ correspond to the gate set introduced in Eqs. (1)-(3). At each RL time step t, the circuit is used to compute a cost function $C(\boldsymbol{\alpha})$ based on the fidelity – see Eq. (4) – either through classical simulation or the Hilbert-Schmidt test (c) – see Eq. (13) – experimentally. The continuous optimizer (d) then outputs a guess for the optimal parameters $\boldsymbol{\alpha}^* = \text{argmin}_{\boldsymbol{\alpha}} [C(\boldsymbol{\alpha})]$ that minimize the cost function for a specific circuit V_t. The minimal value of the cost function is used to assign a reward to the PS agent – see Eq.(29) –, thus closing the RL training loop.

generally considered more challenging than with just two-qubit entangling gates and single qubit rotations [52]. A standard strategy is to progressively increase the number of entangling MS gates on the circuit, accompanied by a suitable number of single and multi-qubit rotations, and to progressively optimize the gate parameters until an acceptable threshold of the figure of merit is reached. This is a viable strategy to obtain one solution for a quantum compilation problem on a trapped-ion device; it is however sub-optimal with respect to the number of gates required. More efficient solutions exist, but the optimization landscape may be difficult to navigate for various algorithms [57]. In particular, as the optimization landscape with respect to the gate angles is particularly vast and depends on the arrangement of the gates on the circuit, it is often necessary to search through several combinations of gate sequences. Random or automated search can be implemented to reduce the number of gates present on the circuit by arbitrarily trying several configurations [24, 52].

3 Methods

In the following section, we discuss our approach to address the three relevant aspects that characterize a circuit synthesis task: the efficient computer-aided simulation of the relevant trapped-ion gates, the optimization of the continuous gate parameters and the optimal arrangement of the gates on the circuit. In addition, we address the possible implementation of our hybrid RL-gradient based optimization method on real quantum devices.

3.1 Dynamics via fast exponentiation

In simulation, direct computation of the gates in Eqs. (1)-(2) is commonly done via direct matrix exponentiation of a Hamiltonian, which is generally slow for large matrices, since it requires several matrix multiplications, each one with a practical complexity of $O(d^{2.8})$ [58], where d is the matrix dimension. Instead, here we show how to significantly reduce the per-iteration computational cost of the ma-

trix exponentiation by factorizing it with respect to the rotation angle θ and to the phase angle ϕ. This is achieved via a spectral decomposition where the phase-independent part can then be cached before the optimization.

Mathematically, the factorization of the matrix exponential for a MS or C_{xy} gate can be written via spectral decomposition as

$$U_H(\theta,\phi) = \exp\{iH(\theta,\phi)\} = \tag{5}$$
$$= V(\theta,\phi)\left(\sum_{l=0}^{d-1} e^{-i\lambda_l(\theta,\phi)}\,|l\rangle\langle l|\right)V^\dagger(\theta,\phi),$$

where $H(\theta,\phi)$ is a θ- and ϕ-dependent Hamiltonian – see exponents in Eqs. (1)-(2) – and V is the matrix of eigenvectors with respective eigenvalues λ_l.

We regroup in terms of degenerate eigenvalues and consider the case where the set of eigenvalues are ϕ-independent, i.e., $\lambda_l = \lambda_l(\theta)$, while a ϕ-dependent unitary is applied to the Hamiltonian, i.e., $H(\theta,\phi) = V(\phi)H(\theta)V^\dagger(\phi)$. As a consequence, the eigenvectors matrix $V = V(\phi)$ is independent of θ, and we can rewrite

$$U_H(\theta,\phi) = \sum_{k=0}^{n_\lambda} V(\phi)\left(e^{-i\lambda_k(\theta)}\sum_{l_k}|l_k\rangle\langle l_k|\right)V^\dagger(\phi) \tag{6}$$

$$= P(\phi)\odot\sum_{k=0}^{n_\lambda} e^{-i\lambda_k(\theta)}D_l, \tag{7}$$

where n_λ is the number of distinct eigenvalues, $|l_k\rangle$ are the respective eigenvectors, $D_k = V(0)\sum_{l_k}|l_k\rangle\langle l_k|\,V^\dagger(0)$ and $P(\phi)$ is a matrix of phase components. In our case, the columns of $P(\phi)$ are all equal and are given by the vector $\boldsymbol{p}(\phi) = \begin{pmatrix}1\\ e^{i\phi}\end{pmatrix}^{\otimes n}$, while $\odot$ represents element-wise (Hadamard) matrix multiplication. The C_{xy} and MS gates have few unique eigenvalues with $\lambda_k = (2k-n)\theta,\ 0\le k\le n_\lambda = n$ and $\lambda_k = (2k-n)^2\theta,\ 0\le k\le n_\lambda = \lceil n+\frac{1}{2}\rceil$, respectively, making the computation with cached D_k particularly efficient. A detailed derivation, together with a discussion of the computational speedup of this representation, is available in Appendix A.

3.2 Continuous gradient-based optimization

We consider the optimization of an observable with respect to the continuous parameters. Although single problems can be optimized effectively by implementing an appropriate discretization method, in general this approach can lead to a sub-optimal solution, since it introduces discontinuities in the search space. Instead, we want to consider the dependency of the fidelity from continuous parameters and calculate its gradient.

3.2.1 Cost function based on known target unitaries.

We consider a quantum circuit composed of a sequence of continuous gates with angle parameters $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1,...,\boldsymbol{\alpha}_L)$ and where each gate is composed of multiple rotation angles $\boldsymbol{\alpha}_1\in\mathbb{R}^{M_1},\dots,\boldsymbol{\alpha}_L\in\mathbb{R}^{M_L}$ and $M_1,...,M_L\ge 1$ and $M=\sum_{m=1}^{L}M_m$. The circuit is then given by

$$V(\boldsymbol{\alpha}) = \prod_{l=1}^{L} V_l(\boldsymbol{\alpha}_l), \tag{8}$$

where $V_l(\boldsymbol{\alpha}_l)$ is an arbitrary parametric unitary with parameters $\boldsymbol{\alpha}_l$.

The gradient with respect to a figure of merit, here the average gate fidelity – see Eq. (4) –, can also be directly computed and are given by

$$\nabla_{\boldsymbol{\alpha}_l}F = \frac{2}{d^2}\,\mathrm{Re}\big\{\mathrm{tr}\big(\nabla_{\boldsymbol{\alpha}_l}V(\boldsymbol{\alpha})U^\dagger\big)\,\mathrm{tr}\big(V(\boldsymbol{\alpha})^\dagger U\big)\big\}, \tag{9}$$

with

$$\nabla_{\boldsymbol{\alpha}_l}V(\boldsymbol{\alpha}) = V_1(\boldsymbol{\alpha}_{l1})\cdots\nabla_{\boldsymbol{\alpha}_l}V_l(\boldsymbol{\alpha}_l)\cdots V_L(\boldsymbol{\alpha}_L), \tag{10}$$

for $1\le l\le L$. The naive element-wise computation of the gradient components uses $M\cdot L$ matrix multiplications, as the matrix in Eq. (9) needs to be evaluated M times and is given by the product of L unitaries. However, the gradient can be computed recursively, a method which is often referred to as GRAPE [59], by storing the values of the intermediate unitaries $W_l = W_{l-1}V_l^\dagger(\boldsymbol{\alpha}_l), W_0 = I_d$ for $l = 1,...,L$ in the product

$$\nabla_{\boldsymbol{\alpha}_l}F = \frac{2}{d^2}\,\mathrm{Re}\Big\{\mathrm{tr}\Big(W_{l-1}\nabla_{\boldsymbol{\alpha}_l}V_l(\boldsymbol{\alpha}_l)W_l^\dagger VU^\dagger\Big)\,\mathrm{tr}\big(V^\dagger U\big)\Big\}. \tag{11}$$

The gradient computation method given in Eq. (11) computes first the intermediate unitaries, for which we need L matrix multiplications and uses them to evaluate the gradient, which compared to Eq. (9) needs only $4M+L+1$ matrix multiplications and therefore scales linearly with the number of gates and gate parameters in the circuit.

Eq. (11) allows us to compute the gradient of any cost function that resembles the structure of Eq. (4) efficiently in numerical simulations. This can be further sped up by similarly analytically computing the Hessian matrix of the cost function [60] or by using GPU or TPU architectures [61]. Eq. (11) can also be useful in an experimental setting when the target dynamics, such as a desired state or unitary evolution, are known a priori.

3.2.2 Cost function based on black-box access to a target unitary.

In many NISQ-relevant algorithms, the structure of the quantum circuit need not be prescriptive. Instead,

we optimize a general circuit ansatz variationally to minimize some performance cost function. In this setting, we once again may optimize both the discrete and continuous degrees of freedom of the circuit to minimize the cost function. One can conceive of situations where the target circuit unitary U may be given as a black-box process [62, 63], that is where any decomposition of the circuit is unknown and we do not have a classical description of the unitary entries, or a circuit to compute them. In this situation, the cost function computing the distance between the black-box target U and a given parametric quantum circuit $V(\boldsymbol{\alpha})$ needs to be estimated.

A possible way to compute distances between a unitary implemented on a quantum computer and a black-box unitary implemented by a different quantum system is the Hilbert-Schmidt test (HST) [64], a generalization of the SWAP test for state preparation, which evaluates the gate fidelity of Eq. (4) on a quantum circuit. A related cost function for black-box quantum compilation is given by the local Hilbert-Schmidt (LHS) test – see Fig. 1, panel (c) –, which has been shown to be easier to optimize and less sensitive to barren plateaus [65, 66]. Observe that the cost function $C_{\mathrm{LHS}}(V,U)$ is bound from above and below by the average gate fidelity given in Eq. (4),

$$C_{\mathrm{LHS}}(V,U) < C_{\mathrm{HST}}(V,U) < nC_{\mathrm{LHS}}(V,U), \qquad (12)$$

which implies that minimizing $C_{\mathrm{LHS}}(V,U)$ also minimizes $C_{\mathrm{HST}}(V,U)$. Other cost functions have been proposed which are based on so-called incoherent learning, i.e., where the quantum computer and the black-box quantum system do not need to interact coherently [67].

For gradient-based optimization, we need to differentiate the cost functions C_{HST} or C_{LHS} with respect to continuous gate parameters. Unfortunately, the gradient method given in Eq. (11) requires access to the intermediate unitaries, which are generally not available in real-world scenarios. Moreover, we do not generally have direct access to the gradient of the unitary operations. As a consequence, we need to use the cost function itself to estimate its gradient with respect to the continuous parameters, which proves slower. In fact, computing the gradient in Eq. (9) with the method of finite differences requires $2M$ evaluations of the test circuit, where M is the total number of parameters. However, finite-difference methods applied to quantum circuits tend to have small signal-to-noise ratios and therefore require large numbers of shots [68], especially when compared to sampling strategies that estimate the gradient via trigonometric interpolation [69, 70].

Let us now consider the specific case of trapped ions with the gate set introduced in Section 2. There are three different types of gates (Z, MS and C_{xy}), with four types of parameters shifts: the three rotation angles of the Z gate, the MS gate and the C_{xy} gate, respectively, and the phase term ϕ of the MS and C_{xy} gates, which is the same for both unitaries (see also Appendix B).

In the following, we consider a cost function on the Hilbert Schmidt test, but the results can be applied to the local test as well. In the case of the Z gate, only one parameter θ is present, whereas for the other two gates we have two possible parameter-shifts. The experimental quantum cost function C_{HST} comparing the parameterized circuit $V(\boldsymbol{\alpha})$ in Eq. (8) with a target unitary U is given by

$$C_{HST}(\boldsymbol{\alpha}) = 1 - \mathrm{Tr}\{H(U \otimes V(\boldsymbol{\alpha})^*)\rho(U^\dagger \otimes V(\boldsymbol{\alpha})^T)\}, \qquad (13)$$

where ρ and H are projectors defined in Appendix B and Ref. [64]. We consider here only the shift with respect to one gate at a time, which we name $V_1(\boldsymbol{\alpha}_1)$, while the remaining gates in the circuit, i.e., $V_2(\boldsymbol{\alpha}_2), ..., V_L(\boldsymbol{\alpha}_L)$, are fixed. This procedure can be repeated for each gate parameter. We first consider the case where $V_1(\boldsymbol{\alpha}_1 = \theta) = Z(\theta)$. The exact derivative of $C_{\mathrm{HST}}(\theta)$ is given by:

$$\frac{\partial}{\partial\theta}C_{\mathrm{HST}}(\theta, Z) = C_{\mathrm{HST}}(\theta + \frac{\pi}{2}, Z) - C_{\mathrm{HST}}(\theta - \frac{\pi}{2}, Z) \qquad (14)$$

For $V_1(\boldsymbol{\alpha}_1 = (\theta,\phi)^T) = C_{xy}(\theta,\phi)$ and following Ref. [69], the parameter derivative with respect to θ is given by

$$\frac{\partial}{\partial\theta}C_{\mathrm{HST}}(\theta,\phi,C_{xy})\Big|_{\theta=0} = \qquad (15)$$
$$\sum_{l=1}^{2n} \frac{(-1)^{l-1}}{2\sin\left(\frac{2l-1}{2n}\pi\right)} C_{\mathrm{HST}}\left(\theta + \frac{2l-1}{2n}\pi, \phi, C_{xy}\right),$$

whereas for $V_1(\boldsymbol{\alpha}_1 = (\theta,\phi)^T) = \mathrm{MS}(\theta,\phi)$, the parameter-shift rule with respect to θ becomes,

$$\frac{\partial}{\partial\theta}C_{\mathrm{HST}}(\theta,\phi,\mathrm{MS})\Big|_{\theta=0} = \qquad (16)$$
$$\sum_{l=1}^{2\lceil\frac{n}{2}\rceil} \frac{(-1)^{l-1}}{2\sin\left(\frac{2l-1}{\lfloor\frac{n}{2}+1\rfloor}\pi\right)} C_{\mathrm{HST}}\left(\theta + \frac{2l-1}{\lfloor\frac{n}{2}+1\rfloor}\pi, \phi, \mathrm{MS}\right)\delta_i^{\mathrm{floor}}.$$

A more simplified rule can be derived for the parameter ϕ, which is valid for both the C_{xy} and the MS gate

$$\frac{\partial}{\partial\phi}C_{\mathrm{HST}}(\theta,\phi) = C_{\mathrm{HST}}(\theta,\phi+\frac{\pi}{4}) - C_{\mathrm{HST}}(\theta,\phi-\frac{\pi}{4}) + \qquad (17)$$
$$+\left(\frac{1}{\sqrt{2}}-\frac{1}{2}\right)C_{\mathrm{HST}}(\theta,\phi-\frac{\pi}{2}) + \left(\frac{1}{\sqrt{2}}+\frac{1}{2}\right)C_{\mathrm{HST}}(\theta,\phi+\frac{\pi}{2}).$$

Using the expressions given by Eq. (14) for the Z gate, by Eq. (15) and Eq. (17) for the C_{xy} gate

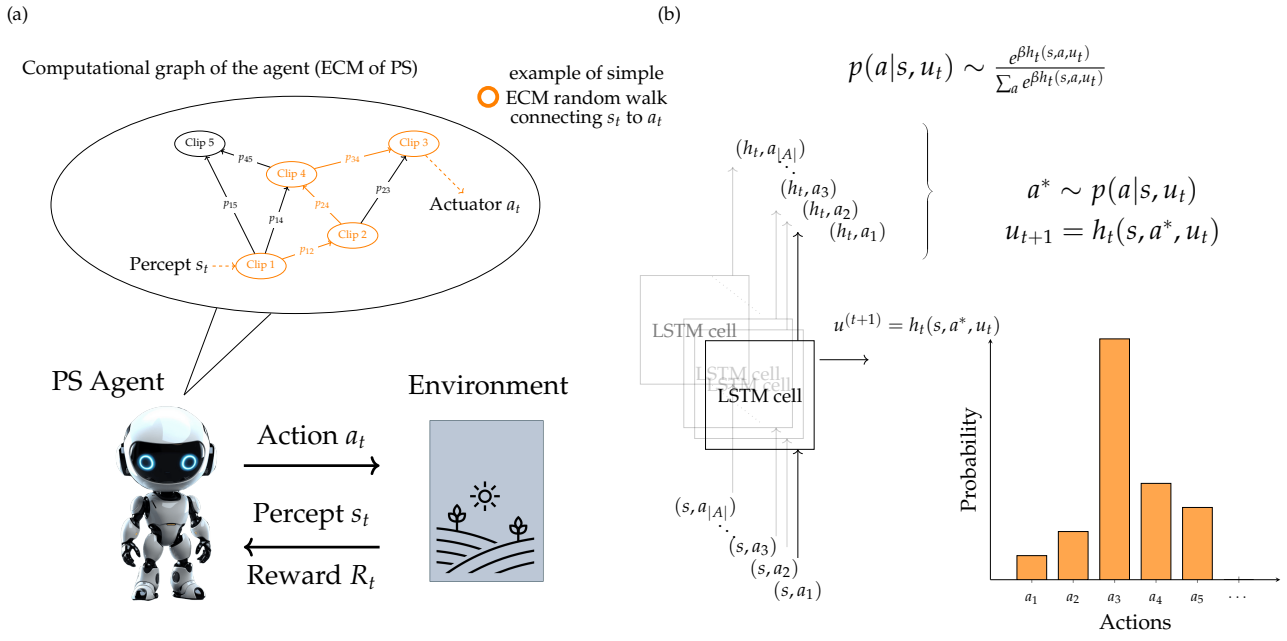


Figure 2: (a) Schematic representation of a PS-environment interaction: the agent is equipped with a memory structure that allows it to process the information input by the environment (in the case of PS, the episodic and compositional memory, ECM [34, 35]). Every perceptual input from the environment triggers a sequence of transitions – showed in orange – within the internal computational graph of the agent and governed by transition probabilities p_{ij}. The sequence of transitions connects a percept clip s_t, which in our case corresponds to the sequence of actions $a_1, ..., a_{t-1}$ used to generate the circuit V_t using the corresponding gates, to an action clip corresponding to a_t. (b) Policy parametrization of the PS-LSTM algorithm. In this implementation, the agent receives a sequential input at time t and constructs a policy by outputting weights $h_t(s,a,u_t)$ corresponding to each action a, the current percept s and the hidden state of the LSTM network u_t. Afterwards, an action a^* is sampled from the policy constructed this way and the weight corresponding to this action is propagated further in the hidden state of the LSTM network, as given in Eq. (23). The weights are then reset when the episode terminates.

and Eq. (16) and Eq. (17) for the MS gate, we can compute any gradient of cost functions estimated in experiments on quantum devices that use cost functions such as Eq. (13). For more details about the parameter-shift rules of the C_{xy} and MS gates, and further possible simplifications, see Appendix B.

3.3 Combinatorial Optimization

In this section, we consider the problem of determining an optimal arrangement of the gates on the quantum circuit. This problem arises from the necessity of minimizing the depth of a quantum circuit to reduce the total circuit execution time, thereby reducing decoherence. And while the circuit can be compiled in layers [71], it is difficult to determine the optimal size due to the large number of possible optimal parameter configurations that produce the same circuit. Therefore, it is necessary to search through various combinations of gate arrangements to determine a minimal one. This task, which partially falls in the realm of combinatorial optimization [72], is particularly suitable for reinforcement learning algorithms [24].

3.3.1 Reinforcement Learning with Projective Simulation

Reinforcement learning describes a class of algorithms which use an agent-environment interaction model to maximize a reward function. In particular, the agent can be represented by a parametrized model, called policy, that outputs action signals on the environment upon receiving observations as inputs. For each action or sequence thereof, the environment returns observations and reward signals. In gradient-based methods, the policy parameters are updated in the direction that maximizes the discounted future expected return [31]. Based upon the different tasks considered, a vast range of algorithms and methods have been developed to tackle various environments [73].

In the following section, we consider the PS architecture – see Fig. 2 (a).

Projective Simulations (PS) is a framework for agency and decision making that has also found applications as a RL agent [74, 36]. In that context, a PS agent interacts with an environment by performing actions sampled from an action space A, whereas the environment provides the agent with perceptual inputs, which reside in a percept space S, and reward

signals. Its central feature is represented by a so-called episodic and compositional memory (ECM), see Fig. 2 (a), a graphical model consisting of a network, or weighted graph, where the vertices are clips and represent, e.g., remembered percepts or remembered actions, but also more general states of the agent's memory. In this framework, the agent creates a clip inside the ECM each time it receives a previously unknown input from the environment, or each time it creates a new action clip or a more abstract clip, e.g., through action composition. This makes it is possible to create ECM networks with complex graph structures, allowing for generalization capabilities [35, 44]. Each input triggers a random walk between the nodes of the ECM that is governed by the edge weights of the graph. We assume in the following that the ECMs created are two-layered networks, with one layer representing percepts clips and the other action clips and where a percept at time t of the RL interaction is connected directly to all action clips. The edge weights are initialized uniformly:

$$\forall a \in A, \forall s \in S : h_0(s,a) = 1. \tag{18}$$

We define the probability distribution[1] over percepts $s \in S$ and actions $a \in A$ as

$$p(a|s) = \frac{e^{\beta h_t(s,a)}}{\sum_a e^{\beta h_t(s,a)}}, \tag{19}$$

for a edge weight $h_t(s,a)$ connecting s to a. The PS algorithm optimizes the policy, similar to other RL frameworks, through a reward signal, which is provided by the environment to the agent. At each RL time-step t, the update rule is given by

$$h_{t+1}(s,a) = h_t(s,a) - \gamma(h_t(s,a) - 1) + \\ + g_t(s,a) R_t, \tag{20}$$

where R_t is the reward at time step t, γ is a damping coefficient that regularizes the result and reduces potential instances of trapping in local minima, $g_t(s,a)$ are so-called glow values [75], which are initially set to zero

$$\forall a \in A, \forall s \in S : g_0(s,a) = 0. \tag{21}$$

They are set (or reset) to $g_t(s,a) = 1$ at time t if the corresponding edge is traversed and decay in value for each time step where they are not used according to the rule

$$g_{t+1}(s,a) = (1-\eta) g_t(s,a), \tag{22}$$

where $0 \leq \eta \leq 1$. The glow mechanism helps distribute the rewards along the entire chain of state-action transitions traversed by the agent in an episode. We see that the edge weights that are rewarded positively grow, thereby enhancing the probability that the same action is chosen again in the future upon receiving a similar percept.

PS has been successfully extended to include more powerful computational structures, such as deep feed-forward energy-based networks [41, 43] and recurrent networks [42]. The aforementioned architectures enable the PS agent to update the policy using function approximators. This allows the agent, as it is the case for deep Q-learning [76], to construct more powerful representations of the percept- and action-spaces and achieve high performances in environments with, e.g., continuous parametric percepts and actions without the need of discretization strategies [31]. We focus here on the Long-Short Memory Network (LSTM) architecture [77, 78], a recurrent neural network that is equipped with an internal memory architecture that helps it learn long-range correlations in a sequential input. In this case, the action sampled at time t also depends on the LSTM internal state u_t, i.e.,

$$a^* \sim p(a|s,u_t) = \frac{e^{\beta h_t(s,a,u_t)}}{\sum_a e^{\beta h_t(s,a,u_t)}} \tag{23}$$

with $s \in S$ and $a, a^* \in A$ and the u-value is updated w.r.t. the sampled action as follows:

$$u_{t+1} = h_t(s, a^*, u_t). \tag{24}$$

The term $u_t = u_t(s,a)$ also depends on percepts and actions, but as we see in Eq. (24), only the u-value corresponding to the action sampled by the agent at time t are propagated as information to the next RL time step, as shown in Fig. 2 (b). This value is the one that carries relevant information about the correlations in the sequential structure of the percepts. We would like to stress that the presence of an additional non-linear term in the update rule in Eq. (23) induces a different type of ECM structure, where the term $u_t(s,a)$ represents an additional edge weight. The update of the reward mechanism can be generalized starting from the standard PS reward update mechanism to fit the training of neural network-based policies, in analogy with the case of Q-learning [41, 42].

4 Circuit compilation

Quantum compilation is the general task of reproducing a general operation $\mathcal{M} \in \mathbb{C}^{4^n \times 4^n}$ on a n-qubit quantum processor. In particular, we consider the compilation of unitary operations $U \in U(n)$.

4.0.1 Layer-based compilation and heuristic search

A general approach for gate synthesis in trapped-ion circuits is discussed in Ref. [52]. This approach is based on the universality of two-qubit entangling

[1] In standard PS, the probability is defined without the exponential factors.

gates for quantum computation [79]. As a consequence, it is possible to compile a quantum algorithm in growing layers, i.e., by iteratively placing entangling MS gates on the circuit followed in each case by a collective rotation of the following type

$$\mathcal{R}(\theta_k, \phi_k, ..., \theta_{k+n+1}, \phi_{k+1}) = \tag{25}$$
$$= C_{xy}(\theta_k, \phi_k) \prod_{i=1}^{n} Z_i(\theta_{k+i}) C_{xy}(\theta_{k+n+1}, \phi_{k+1}),$$

where the gates C_{xy} and Z are defined in Eqs. (2)-(3). For each layer of gates present on the circuit, we have one MS gate and one general rotation $\mathcal{R}$ – see Eq. (25) –, which consists of n local Z gates and two C_{xy} gates.

Algorithm 1 Exhaustive search with layer-based compilation

Input Target unitary U, threshold ϵ, cost function C.

Output $V^*, \boldsymbol{\alpha}^*$.

▷ Construct layers:
for $L = 1$ to L_{MS} **do**
  ▷ Add MS gate and rotations:
  $V(\boldsymbol{\alpha}) = \prod_{l=1}^{L} V_l(\boldsymbol{\alpha}_l)$
  ▷ All angles but the ones of MS gates:
  $\boldsymbol{\alpha} = (\theta_1, \phi_1, ..., \theta_{(n+2)L}, \phi_{2L})^T$
  $\tilde{K} = (n+2)(L+1)$
  ▷ Loop over numbers of rotations:
  for $k = 1$ to $\tilde{K}$ **do**
    ▷ Loop over combinations of indices:
    for $j = 1$ to $\binom{\tilde{K}}{k}$ **do**
      ▷ Set angles with chosen indices to zero:
      $\alpha_{\sigma(j,k)} = 0$
      $\tilde{\boldsymbol{\alpha}}^k = (..., \alpha_{\sigma(j,k)} = 0..., \alpha_{\sigma(j,k)} = 0, ...)^T$
      ▷ Cost function according to Eq. (27):
      $C(\tilde{\boldsymbol{\alpha}}^k) = 1 - F(V, U)$
      $C^* = \min(C(\tilde{\boldsymbol{\alpha}}^k), \nabla_{\boldsymbol{\alpha}^k} C(\tilde{\boldsymbol{\alpha}}^k))$
      if $C^* \leq \epsilon$ **then**
        $\boldsymbol{\alpha}^* = (\tilde{\boldsymbol{\alpha}}^k)^*$
        $V^* = V$
        break
      else
        continue
      end if
    end for
  end for
end for

The unitary added to the circuit at each layer l is:

$$V_l(\boldsymbol{\alpha_l}) = \text{MS}(\theta_l^{\text{MS}}, \phi_l^{\text{MS}})\ \mathcal{R}(\theta_l, \phi_l, ..., \theta_{l+n+1}, \phi_{l+1}). \tag{26}$$

Algorithm 2 PS-based compilation

Input Target unitary U, Action set $G_1 = \text{MS}, G_2 = C_{xy}, G_3 = Z_1, ..., G_{n+2} = Z_n$, threshold function ϵ_t, cost function C

Output $V^*, \boldsymbol{\alpha}^*$

for $e = 1$ to $E_{\max}$ **do**
  $s_1 = (0, ..., 0), V_1 = I_d$
  for $t = 1$ to $L_{\max}$ **do**
    ▷ Sample action according to Eq. (19):
    $a_t \sim \pi(a|s_t)$
    $V_{t+1}(\boldsymbol{\alpha}_t) = G_{a_t} V_t(\boldsymbol{\alpha}_{t-1})$
    $s_{t+1} = (a_1, ..., a_t, 0, ..., 0)$
    $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_t) \sim 2\pi \cdot \mathcal{N}(\mathbf{0}_{\boldsymbol{\alpha}}, \boldsymbol{I}_{\boldsymbol{\alpha}})$
    ▷ According to Eq. (27):
    $\boldsymbol{\alpha}^* = \text{argmin}(C(\boldsymbol{\alpha}), \nabla_{\boldsymbol{\alpha}} C(\boldsymbol{\alpha}))$
    $R_t = \begin{cases} 2 & \text{if } \epsilon_{\min} \leq C(\boldsymbol{\alpha}^*) \leq \epsilon_t \\ 10 & \text{if } C(\boldsymbol{\alpha}^*) \leq \epsilon_{\min} \\ 0 & \text{otherwise} \end{cases}$
    if $C^* \leq \epsilon_t$ **then**
      $V_* = V_{t+1}$
      break
    end if
    Update rule for all $h_{t+1}(s_t, a_t)$ (see Eq. (20))
  end for
  Update threshold ϵ_t according to Eq. (30)
end for

The cost function

$$C(\boldsymbol{\alpha}) = 1 - \frac{1}{d^2} \left| \text{tr} \left\{ \prod_{l=1}^{L} V(\boldsymbol{\alpha}_l) U^\dagger \right\} \right|^2, \tag{27}$$

based on the gate fidelity in Eq. (4), has to be minimized with respect to the angle parameters $\boldsymbol{\alpha}$. After obtaining the optimal parameters $\boldsymbol{\alpha}^*$, if the desired error threshold is reached, the algorithm terminates, otherwise a new layer of gates is placed on the circuit, up to a maximum number of layers L_{MS}. By running this procedure iteratively with different angle parameter sets, we can search through the optimization landscape to find different gate decompositions (see Algorithm 1 and Ref. [52]).

However, while this method can be applied to circuits with a small number of layers and qubits, its use becomes impractical as these two parameters grow. In fact, the number of total combinations to analyze for

a given number of MS layers L_{MS} and a number of qubits n is

$$N_{\text{combinations}} = \frac{2^{(n+2)(L_{\text{MS}}+1)} - 1}{1 - \frac{1}{4}^{(n+2)}} - L_{\text{MS}}^2. \quad (28)$$

For a 3-qubit circuit with L layers of entangling gates, the number of possible circuits is $N_{\text{combinations}} = 1049591$. For a 4-qubit circuit with 5 entangling gates the number of combinations is approximately $N_{\text{combinations}} \sim 2^{36}$. We see that already for 4-qubit gates, a brute force approach is unfeasible. Moreover, due to the large search space, even approaches based on random search are not a viable option, especially if the number of entangling gates is large.
In the following section, we suggest a different approach to the optimization scheme where the position of the discrete parameters is modified by a RL agent equipped with a curriculum scheme, whereas the continuous parameters are optimized at each iteration and using a pre-defined heuristic for angle initialization. This can offer benefits in several situations where the number of gates is large enough to make the use of automated search prohibitive but not so large to make the problem completely intractable from the point of view of continuous parameter optimization.

4.0.2 Reinforcement learning-based compilation

In the following, we discuss the implementation of RL-based compilation. In this system, the agent acts on the environment, which represents a quantum circuit, by choosing a gate from, e.g., the gate set of Section 2 and placing it on the circuit. As an observation, the agent receives information about the environment internal state. This can vary depending on the task to be solved: in Ref. [24], e.g., the input is a single-qubit unitary and the task is to construct a pre-trained optimal compiler that constructs any unitary with minimal average number of actions. In Refs. [42, 80], the agent receives an encoded representation of the quantum circuit in terms of gates and qubits. The circuit-based input has the advantage of scaling linearly with the number of qubits for n-qubit entangling gates and its sequential structure makes it suitable for recurrent or autoregressive policies [30, 27].

The perceptual input of the PS-LSTM agent is given by the current gate on the circuit. The action of the PS-LSTM agent is placing one further gate on the circuit. A representation of the interaction between the agent and the quantum circuit environment is given in Fig. 1: at each time step, the agent – Fig. 1 (a) – can place one or more gates on the circuit. Then the circuit – Fig. 1 (b) – is mapped to the corresponding unitary function, which depends on parameters $\boldsymbol{\alpha}$. The gradient-based optimizer – Fig. 1 (d) –, i.e., the L-BFGS-B algorithm [81], is given the task of finding the optimal set of parameters to maximize the fidelity – Fig. 1 (c) – between the current circuit and a target unitary process. The reward function and the percept for the next interaction step are constructed based on the result of the optimization. The algorithm is shown in Algorithm 2 for the standard PS method and can be easily generalized to a framework with deep networks [41, 42].

Furthermore, one may design different types of RL environments based on the way the agent places gates on the circuit. We develop here two different approaches for two different types of RL environments: In the first environment, we just mimic the structure of layer-based compilation, in which we keep the entangling gates fixed and then let the agent vary the rotations between them. This assumes that the RL interaction is defined by the number of gates placed between each entangling layer and the total amount of entangling layers considered. This architecture has the advantage of restricting the search space of the RL agent, but it allows for less exploration of the cost function landscape.

The second architecture allows the agent to place any available gate, entangling or non-entangling, on the circuit and as such does not restrict the action of the agent on the quantum circuit, with one single exception: if the agent places the same type of gate twice in a row on the circuit, these two gates are merged together in one single gate. This is needed to avoid the agent getting stuck in a loop where it keeps performing the same operation over and over again, without any meaningful exploration of the optimization landscape from the perspective of the continuous optimizer. We employ this architecture in our simulations.

In our implementation, the action a_t chosen at time t is the index of certain gate in the gate set, whereas the corresponding percept s_t is given by the concatenation of previously chosen actions $(a_1, ..., a_{t-1})$. Thus, the action space is given by the gate indices $A = \{1, 2, ..., n+2\}$ and the percept space by the Cartesian product of $L_{\max}$ action spaces: $S = A^{\times L_{\max}}$. The PS-LSTM algorithm, however, can also be given just the action at time $t-1$ as percept, since the LSTM memory can automatically capture the correlations between the elements in the sequence. Here, we adopt a RL training procedure with a curriculum scheme as described in Ref. [26]. This allows the agent to sufficiently explore the solution space and gradually adapt the solution. More specifically, for each task of quantum circuit optimization, we define a curriculum strategy where we reward the agent at time step t each time it finds a sequence with achieved minimal infidelity $C(\boldsymbol{\alpha}^*)$ lower than a chosen moving threshold ϵ_t and a fixed threshold $\epsilon_{\min} = 10^{-2}$:

$$R_t = \begin{cases} 2 & \text{if } \epsilon_{\min} < C(\boldsymbol{\alpha}^*) < \epsilon_t \\ 10 & \text{if } C(\boldsymbol{\alpha}^*) < \epsilon_{\min} \\ 0 & \text{otherwise} \end{cases}. \quad (29)$$

The episode terminates when the reward the cost

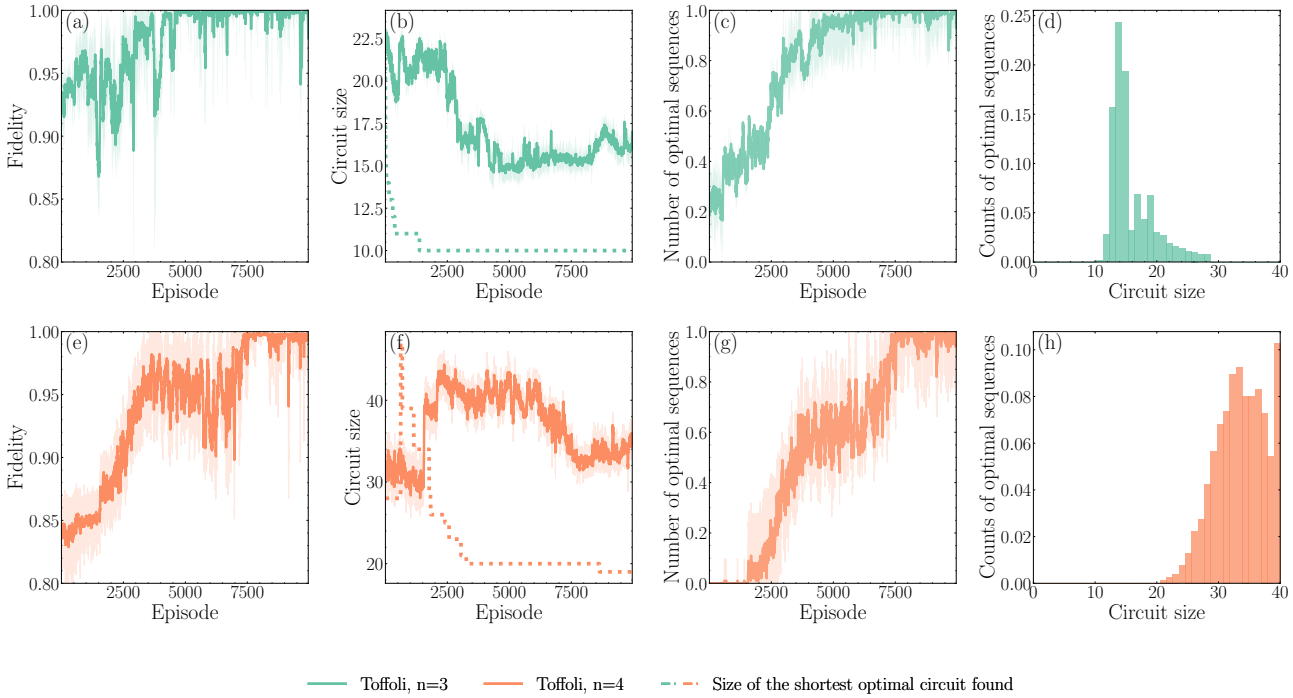


Figure 3: Quantum circuit optimization of different gates using the gate set defined in Eqs. (1)-(3). (a), (e) show the average reward; (b), (f) the average circuit size (thick line) and the size of the best circuit found so far by all agents (dotted line); (c), (g) the number of optimal sequences per episode; (d), (h) are histograms which show the distribution of the optimal gate sequences over the circuit size. All lines refer to simulations of the 3-qubit Toffoli gate, first compiled on a 3-qubit (green line, average over 10 agents and 20 episodes) and then on a 4-qubit circuit (orange line, average over 5 agents and 20 episodes). Due to the n-qubit interaction of the trapped-ion gates, the optimal sequence that generates the gate – which we define as the shortest sequence whose infidelity falls below $\epsilon_{\text{min}} = 10^{-2}$ – increases in size from 10 to 19 gates. Shaded regions in the plots represent the corresponding standard deviations. While the average fidelity increases to reach the maximum for both circuits, we see that the average circuit length appears to be higher than the shortest circuit length. There are possible explanations for this: the shortest sequences can be harder to optimize, so there is a higher chance that the optimizer fails at outputting the cost function minimum for a given circuit structure and the curriculum scheme, that modifies the problem online during the training. Overall, we observe that the size of the optimal circuit decreases as training progresses, thus validating the successful optimization of the policy.

function minimum in a given time step falls below the threshold ϵ_t or when the maximal length of the circuit per episode, L_{max}, is reached. The RL training terminates upon reaching the maximum number of episodes E_{max}. The reward scheme helps to progressively increase the fidelity throughout training without allowing for too long circuits. The threshold is then lowered as episodes progress based on previous rewards obtained by the agent. In our implementation, we lower the threshold when it has been surpassed by the agent at least 500 times using the following scheme:

$$\epsilon_{t+1} = \begin{cases} \epsilon_{\text{min}} + \frac{1}{2}(\epsilon_{\text{min}} - \epsilon_t) & \text{if } \epsilon_{\text{min}} \leq \epsilon_t \leq 1 \\ \epsilon_{\text{min}} & \text{if } \epsilon_t \leq \epsilon_{\text{min}} \\ 1 & \text{otherwise.} \end{cases} \tag{30}$$

5 Results

We test our algorithm on two relevant tasks of circuit optimization: Standard gate compilation, which can be useful in particular for experimental applications of frequently used gates (Toffoli, etc.) and the simulation of black-box unitary processes with quantum circuits. The latter framework is particularly interesting for quantum simulation and offers the possibility of implementing our algorithm directly on an experimental setting where a black-box unitary process has to be simulated by a quantum circuit with available gates.

5.1 Example 1: gate compilation

As a first application of the method presented above, we consider the problem of compiling a gate on an ion-trap quantum processor using the gate set given in Eqs. (1)-(3). For this class of tasks, we choose two gate compilation problems which can be of interest for typical applications, before passing to a more general framework which considers black-box unitaries. First, we consider a standard quantum computing gate, the

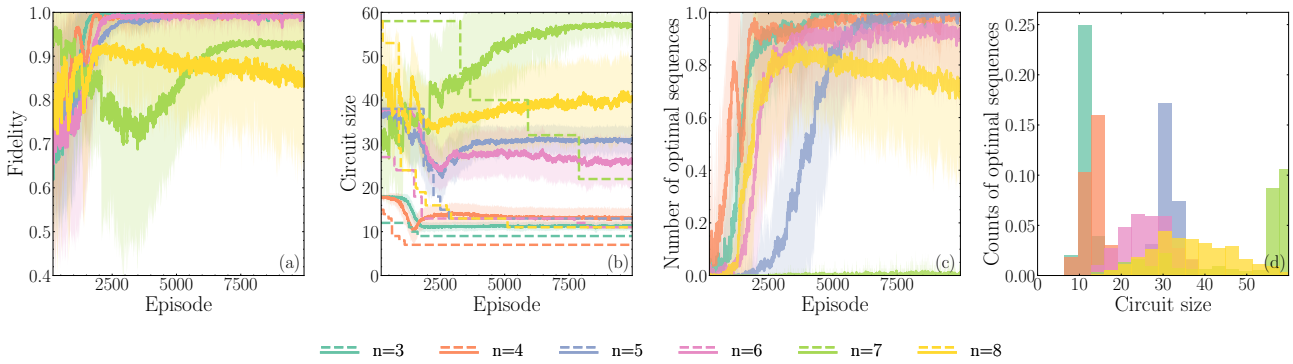


Figure 4: (a) Fidelity; (b) circuit size (thick lines) and size of the best circuit found by all agents (dotted lines); (c) number of optimal sequences per episode and (d) histogram of the optimal sequences – i.e., whose infidelity falls below $\epsilon_{\text{min}} = 10^{-2}$ – based on the circuit size for the UCC operators defined in Eq. (31) for $\beta = \frac{\pi}{2}$ and for 3, 4, 5, 6, 7, 8 qubits. The learning curves show the average over 20 agents, after which an average over a time window of 20 episodes is performed. We observe how the average maximal fidelity reached by the agents decreases as a function of the number of qubits, whereas the optimal circuit size increases, which means that a longer circuit is necessary to synthesize the desired operator. Moreover, the fraction of optimal sequences found by the agent is also decreasing for higher numbers of qubits, as optimal policies become more and more sparse and thus harder to discover for the PS-LSTM agent. The hardest task for the PS-LSTM agent proves to be 7-qubit UCC operator, where the learning curve of the agents fails to converge, while, e.g., for the 8-qubit UCC unitary the agent can find sequences with very high fidelities ($1 - F < 10^{-5}$), even though the convergence is sub-optimal. We also observe that for each learning curve there is a dip at some point in the training: this is most likely due to the curriculum scheme, which modifies the reward threshold during training and therefore influences the size of the optimal circuits found by the agent. However, when considering the ensemble of simulations, we see that the agents are capable of finding shorter and shorter optimal circuits. Shaded regions in the plots represent the corresponding standard deviations.

3-qubit Toffoli gate [82, 83] – see Fig. 3 – for a 3-qubit (green line, upper four plots) and a 4-qubit circuit (orange line, lower four plots). The dashed lines in Fig. 3 show the optimal solutions found by the agent. In the 3-qubit case, this solution matches agrees with the results given in Ref. [52]. Fig. 3 (a), (e) show the fidelity of the sequences produced and optimized by the agents as the learning progresses, (b), (f) show the average circuit size and the size of the shortest circuit found by the agent, (c), (g) show the average number of optimal sequences, i.e., with fidelity higher than 0.99, found in each episode (which in the best-case scenario should converge to one per episode) and (d), (h) show histograms representing the number of sequences for different bins of circuit size. We observe from the first and second plot in each row of Fig. 3 that the agents are capable of maximizing the fidelity and at the same time of reducing the average circuit size in both cases. Moreover, while the average circuit size is larger than the size of the best circuit, most likely due to the presence of the curriculum and the influence of local minima on the angle optimization, we also see that the agents find shorter and shorter optimal circuits as the training progresses, hinting that the at least one agent in the ensemble is indeed learning to optimize the circuit correctly. Moreover, the third plot in each row shows us that the agents find an increasing number of high-fidelity sequences during training. The minimal sequences found by the agents are located in the leftmost tail of the distribution. The 3-qubit Toffoli gate implemented on a 4-qubit circuit could have in principle a relatively long generating quantum circuit in the chosen gate set, since the 4-qubit gate set also affects the qubits left unchanged by the Toffoli gate. We also observe, in our simulations, that sequences generating this gate are sparse in the action space, which could make it generally difficult to produce this gate on a register of n qubits without the help of MS and C_{xy} gates acting only on subspaces of the register. We test whether our method can discover a circuit of reasonable size. For large numbers of qubits and large circuit sizes, we observe that although the algorithm can find shorter circuits, it is still impaired by the computational overhead of simulating such circuits exactly. In this case, the gate decomposition method introduced in Section 3.1 provides a useful speedup for RL simulations, especially if compiled on GPU architectures.

5.2 Example 2: General quantum process simulation of a Hamiltonian model

As a second example, we compile parametric-type operators that play a relevant role in quantum chemistry [84, 85], i.e., UCC-type operators. These are part of a more general class of many-body operators [86]:

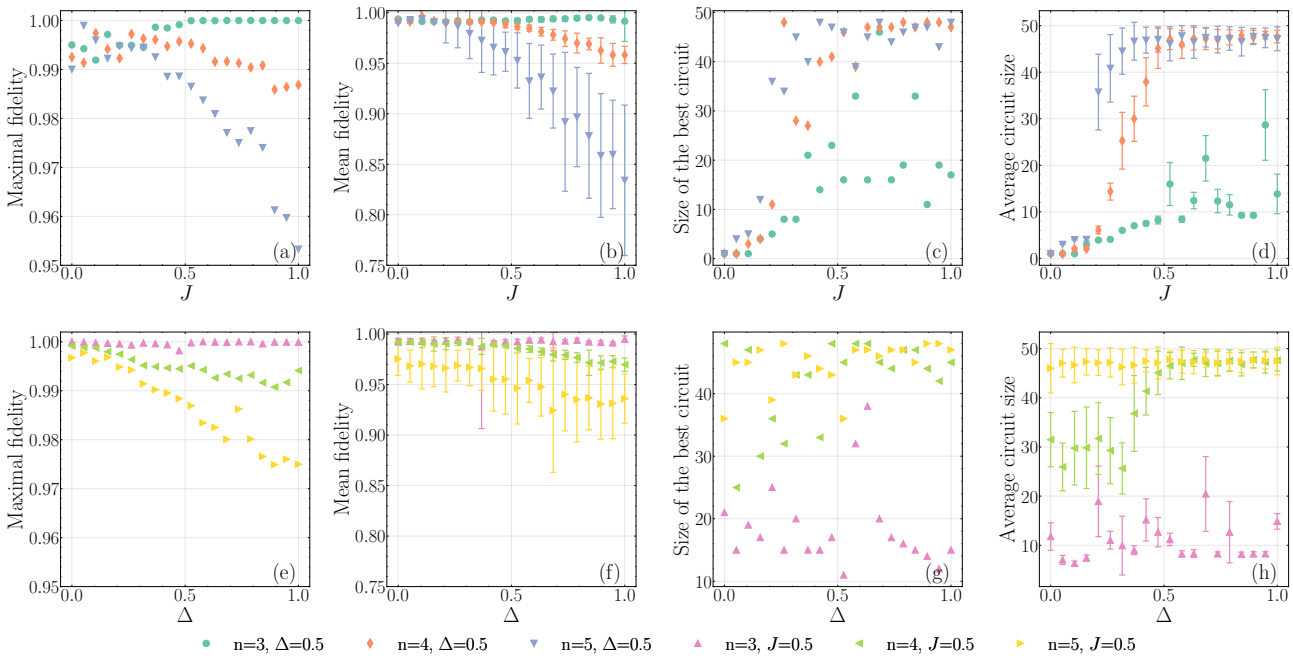


Figure 5: These plots summarize the result of simulating the XXZ-model unitary with our approach for different parameter configurations for $n = 3, 4, 5$ qubits (3 agents for each parameter sample). (a) shows the maximal value of the fidelity found by the agents, (b) the mean fidelity, (c) the size of the shortest circuit associated with the fidelity in (a) and (d) the average circuit size, as a function of the coupling J of the XXZ model for two different configuration of the transverse coupling $\Delta = 0.5$ with varying J and and $J = 0.5$ with varying Δ (dark green, orange and blue and pink, light green and yellow, respectively). The evolution time was fixed to $\tau = 0.25$ and the maximum size of the circuit to 50. The average fidelity is calculated over the last 200 episodes. Vertical bars show the standard error for the mean values of fidelity and circuit size. We see here that the circuit size increases dramatically as the parameters Δ and J increase. We also see from the number of outliers in (c) and (g), that for certain values of the parameters it is hard for the agents to exactly retrieve the optimal circuit.

$$U(\beta) = \exp\left(i\beta\left(\prod_{i=1}^{n}(\sigma_x^{(i)} - i\sigma_y^{(i)}) + \text{h.c.}\right)\right). \quad (31)$$

The class of operators defined by Eq. (31) resembles the collective rotations used in trapped-ion gate sets given in Eqs. (1)-(3), they are however sparser. Due to their importance, they represent our first candidate for process simulation on the quantum circuit. The results for the simulation using PS-LSTM of several such operators – $n = 3, 4, 5, 6, 7, 8$ – and for $\beta = \frac{\pi}{2}$ is given in Fig. 4. We observe that the agent is able to increase the fidelity up to the optimal value. We also see that the size of the optimal circuit generating the corresponding operators increases with the number of qubits. Furthermore, due to the growing number of local rotations available to the agent and the sparseness of the reward, it becomes increasingly difficult to find and reward optimal sequences. This also shows the benefit of implementing curriculum strategies in such hybrid discrete-continuous optimization problems, where the landscape both for the RL agent and for the numeric optimizer become increasingly sparse. We also note that the generation of 7 and 8-qubit UCC operators proves more challenging. In particular, for the 7-qubit UCC unitary, the agent is able to raise the fidelity to values close to $F = 0.99$, but it cannot significantly increase the number of circuits with $F > 0.99$ over the course of the training. In the case of the 8-qubit UCC operator, the agent instead proves capable of discovering sequences with very high fidelity, i.e $1 - F < 10^{-5}$, although the average fidelity worsens slightly towards the end of the training. We also see that an ensemble of agents can successfully reduce the size of the best circuit, even in those cases where the average performance worsens during training.

We would like now to consider a (black-box) unitary $U(\tau) = e^{-iH\tau}$, where τ is the evolution time described by a Hamiltonian of the following type:

$$H = -2h\sum_{i=1}^{n}\sigma_z^{(i)} - J\sum_{i=1}^{n-1}\left(\sigma_x^{(i)}\otimes\sigma_x^{(i+1)} + \sigma_y^{(i)}\otimes\sigma_y^{(i+1)}\right. \quad (32)$$
$$\left. +\Delta\left(\sigma_z^{(i)}\otimes\sigma_z^{(i+1)} - \frac{1}{4}I\right)\right),$$

which is usually referred to as the XXZ model [87].

We want to analyze how the optimal quantum circuit found by the RL agent changes when we vary the Hamiltonian parameters. For the XXZ model, this parameter variation represents for instance the transition from a XX model when $\Delta = 0$ to a XXX model for $\Delta = 1$, or from ZZ interaction for $J = 0$ to a XXZ model for $J > 0$. In fact, we observe in different parameter regions different behaviours of the circuit structure. Results of several RL runs for two different configurations, one with fix coupling $J = 0.5$ and varying Δ and another one with fix $\Delta = 0.5$ and varying J of the XXZ model unitary are shown in Fig. 5 for 3, 4 and 5 qubits. Plots (a), (e) show the maximal fidelity found by the agents for each run, plots (b), (f) the mean fidelity, plots (c), (g) the number of gates in the shortest circuit among those with the highest fidelity and plots (d), (h) the average circuit size, all represented in their functional dependence from the coupling J and transverse coupling Δ of the XXZ model for two different configuration of the transverse coupling $\Delta = 0.5$ with varying J and and $J = 0.5$ with varying Δ (dark green, orange and blue and pink, light green and yellow, respectively). We also see how rapidly the circuit size grows in the presence of entanglement, for example from $J = 0$ to $J = 1$, whereas the increase in average circuit size seems less pronounced as we vary Δ. We observe that the optimal circuit size can also decrease, for example when $\Delta = 1$ and $J = 0.5$. We also see some instabilities and high variance in both the average circuit size and the size of the best circuit discovered, though it is hard to determine whether the agent fails at finding a shorter circuit for a specific parameter or the problem becomes suddenly harder to represent with the given gate set due to the parameter variation.

6 Conclusion

In this work we construct a framework to learn both classically simulated and black-box unitaries on a quantum circuit using RL and unconstrained optimization. Our simulation is specifically tailored to an ion-trap architecture based on collective gates and local rotations. We demonstrate the synthesis of optimal circuits for Toffoli gates and UCC operators for varying numbers of qubits. As instances of black-box unitaries, we also consider Hamiltonian simulations of the XXZ model. More specifically, we study the convergence and the quality of the solutions found by agent and optimizer as we modify relevant parameters of the underlying black-box unitary, such as the coupling in the XXZ Hamiltonian. After testing the algorithm on different unitary process simulation scenarios, we observe that the optimization of circuits is generally possible even for large numbers of gates, it is however difficult to foresee how sparse the cost function minima can be as we increase the number of qubits and reduce the sparsity in the corresponding Hamiltonian. Possible improvements include combining the RL search with a graph traversal algorithm to have a more efficient exploration of the discrete action space [38, 29]. We expect that this approach can be applied to different architectures beyond trapped ions, and more carefully engineered reward functions can be developed to further enhance the discovery of optimal circuits. Our unified optimization, combining circuit compilation and unitary synthesis, may find use both in classical pre-optimization and in experimental circuit learning, be it for in-situ (variational) algorithms or module-compilation tasks.

7 Data and Code availability

The code and the data are available at the following link: https://github.com/franz3105/RL_Ion_gates.

Acknowledgement

We would like to thank Markus Schmitt, Robert Zeier and Marco Canteri for the useful discussions. FP, MS and FM acknowledge support from the German Federal Ministry of Education and Research (BMBF) within the framework programme "Quantum technologies – from basic research to market" (Project QSolid, Grant No. 13N161), the European Union's Horizon Programme (HORIZON-CL4-2021-DIGITALEMERGING-02-10) Grant Agreement 101080085 QCFD and HORIZON-CL4-2022-QUANTUM-02-SGA via the project 101113690 (PASQuanS2.1), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – Cluster of Excellence Matter and Light for Quantum Computing (ML4Q) EXC 2004/1 – 390534, and from the Jülich Supercomputing Center through the JUWELS and JURECA cluster. SJ acknowledges the Austrian Academy of Sciences as a recipient of the DOC Fellowship. This research was funded in whole or in part by the Austrian Science Fund (FWF) through the DK-ALM W1259-N27 ([Grant DOI: 10.55776/W1259] and the SFB BeyondC F7102 [Grant DOI: 10.55776/F71]. For open access purposes, the authors have applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission. This work was also co-funded by the European Union (ERC, QuantAI, Project No. 101055129). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Simulations were realized using the libraries NUMBA [88], JAX [89], PYTORCH [90] and SCIPY [91]. The robot picture

in Fig. 1 and Fig. 2 was generated by the authors using Midjourney.

References

[1] F. Arute, K. Arya, R. Babbush, *et al.*, Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).

[2] J. Preskill, Quantum Computing in the NISQ era and beyond, Quantum **2**, 79 (2018).

[3] C. Bravo-Prieto, R. LaRose, M. Cerezo, *et al.*, Variational Quantum Linear Solver, Quantum **7**, 1188 (2023).

[4] A. Peruzzo, J. McClean, P. Shadbolt, *et al.*, A variational eigenvalue solver on a photonic quantum processor, Nature Communications **5**, 4213 (2014).

[5] M. Cerezo, A. Arrasmith, R. Babbush, *et al.*, Variational quantum algorithms, Nature Reviews Physics **3**, 625 (2021).

[6] F. Kreppel, C. Melzer, D. Olvera Millán, *et al.*, Quantum Circuit Compiler for a Shuttling-Based Trapped-Ion Quantum Computer, Quantum **7**, 1176 (2023).

[7] E. Younis, C. C. Iancu, W. Lavrijsen, *et al.*, Berkeley quantum synthesis toolkit (bqskit) v1 (2021).

[8] Qiskit contributors, Qiskit: An open-source framework for quantum computing (2023).

[9] D. Venturelli, M. Do, B. O'Gorman, *et al.*, Quantum circuit compilation: An emerging application for automated reasoning, in *Scheduling and Planning Applications Workshop* (2019).

[10] R. Wille, S. Hillmich, and L. Burgholzer, Efficient and correct compilation of quantum circuits, in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* (2020) pp. 1–5.

[11] M. Maronese, L. Moro, L. Rocutto, and E. Prati, Quantum compiling, preprint at arXiv:2112.00187 (2021).

[12] A. Sørensen and K. Mølmer, Quantum computation with ions in thermal motion, Physical Review Letters **82**, 1971–1974 (1999).

[13] K. Mølmer and A. Sørensen, Multiparticle entanglement of hot trapped ions, Phys. Rev. Lett. **82**, 1835 (1999).

[14] M. Malekakhlagh, How the Cross Resonance Gate Works, `https://thequantumaviary.blogspot.com/2021/07/how-cross-resonance-gate-works.html` (2021), [Online; accessed 06-January-2022].

[15] P.-L. Dallaire-Demers and F. K. Wilhelm, Quantum gates and architecture for the quantum simulation of the fermi-hubbard model, Phys. Rev. A **94**, 062304 (2016).

[16] F. Preti, T. Calarco, and F. Motzoi, Continuous quantum gate sets and pulse-class meta-optimization, PRX Quantum **3**, 040311 (2022).

[17] M. S. Daoud, M. Shehab, H. M. Al-Mimi, *et al.*, Gradient-based optimizer (gbo): A review, theory, variants, and applications, Archives of Computational Methods in Engineering (2022).

[18] D. Simon, *Evolutionary optimization algorithms* (Wiley-Blackwell, Hoboken, NJ, 2013).

[19] J. A. Nelder and R. Mead, A Simplex Method for Function Minimization, The Computer Journal **7**, 308 (1965).

[20] C. M. Dawson and M. A. Nielsen, The solovay-kitaev algorithm, Quantum Info. Comput. **6**, 81–95 (2006).

[21] F. Vatan and C. Williams, Optimal quantum circuits for general two-qubit gates, Phys. Rev. A **69**, 032315 (2004).

[22] R. Wille, O. Keszocze, M. Walter, *et al.*, Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits, in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)* (2016) pp. 292–297.

[23] M. G. Davis, E. Smith, A. Tudor, *et al.*, Towards optimal topology aware quantum circuit synthesis, in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)* (2020) pp. 223–234.

[24] L. Moro, M. G. A. Paris, M. Restelli, and E. Prati, Quantum compiling by deep reinforcement learning, Communications Physics **4**, 178 (2021).

[25] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, Quantum circuit optimization with deep reinforcement learning, preprint at arXiv:2103.07585 (2021).

[26] M. Ostaszewski, L. M. Trenkwalder, W. Masarczyk, *et al.*, Reinforcement learning for optimization of variational quantum circuit architectures, in *Advances in Neural Information Processing Systems*, Vol. 34, edited by M. Ranzato, A. Beygelzimer, Y. Dauphin, *et al.* (Curran Associates, Inc., 2021) pp. 18182–18194.

[27] V. Sivak, A. Eickbusch, H. Liu, *et al.*, Model-free quantum control with reinforcement learning, Physical Review X **12** (2022).

[28] S. Borah, B. Sarma, M. Kewming, *et al.*, Measurement-based feedback quantum control with deep reinforcement learning for a double-well nonlinear potential, Phys. Rev. Lett. **127**, 190403 (2021).

[29] Y.-H. Zhang, P.-L. Zheng, Y. Zhang, and D.-L. Deng, Topological quantum compiling with reinforcement learning, Phys. Rev. Lett. **125**, 170501 (2020).

[30] J. Yao, L. Lin, and M. Bukov, Reinforcement learning for many-body ground-state preparation inspired by counterdiabatic driving, Phys. Rev. X **11**, 031070 (2021).

[31] R. S. Sutton and A. G. Barto, *Reinforcement*

Learning: An Introduction, 2nd ed. (The MIT Press, 2018).

[32] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, Mastering the game of go without human knowledge, Nature **550**, 354 (2017).

[33] J. Jumper, R. Evans, A. Pritzel, *et al.*, Highly accurate protein structure prediction with alphafold, Nature **596**, 583 (2021).

[34] H. J. Briegel and G. De las Cuevas, Projective simulation for artificial intelligence, Scientific Reports **2**, 400 (2012).

[35] A. A. Melnikov, A. Makmal, V. Dunjko, and H. J. Briegel, Projective simulation with generalization, Scientific Reports **7**, 14430 (2017).

[36] A. A. Melnikov, H. Poulsen Nautrup, M. Krenn, *et al.*, Active learning machine learns to create new quantum experiments, Proceedings of the National Academy of Sciences **115**, 1221 (2018).

[37] J. Wallnöfer, A. A. Melnikov, W. Dür, and H. J. Briegel, Machine learning for long-distance quantum communication, PRX Quantum **1**, 010301 (2020).

[38] M. Dalgaard, F. Motzoi, J. J. Sørensen, and J. Sherson, Global optimization of quantum dynamics with alphazero deep exploration, npj Quantum Information **6**, 6 (2020).

[39] H. P. Nautrup, N. Delfosse, V. Dunjko, *et al.*, Optimizing Quantum Error Correction Codes with Reinforcement Learning, Quantum **3**, 215 (2019).

[40] K. Ried, T. Müller, and H. J. Briegel, Modelling collective motion based on the principle of agency: General framework and the case of marching locusts, PLOS ONE **14**, 1 (2019).

[41] S. Jerbi, L. M. Trenkwalder, H. P. Nautrup, *et al.*, Quantum enhancements for deep reinforcement learning in large spaces, PRX Quantum **2** (2021).

[42] F. Preti, Deep projective simulation and state preparation, master thesis (2020).

[43] H. P. Nautrup, T. Metger, R. Iten, *et al.*, Operationally meaningful representations of physical systems in neural networks, Machine Learning: Science and Technology **3**, 045025 (2022).

[44] B. Eva, K. Ried, T. Müller, and H. J. Briegel, How a minimal learning agent can infer the existence of unobserved variables in a complex environment, Minds and Machines (2022).

[45] S. Kwon, A. Tomonaga, G. Lakshmi Bhai, *et al.*, Gate-based superconducting quantum computing, Journal of Applied Physics **129**, 041102 (2021).

[46] M. Saffman, Quantum computing with atomic qubits and rydberg interactions: progress and challenges, Journal of Physics B: Atomic, Molecular and Optical Physics **49**, 202001 (2016).

[47] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, Trapped-ion quantum computing: Progress and challenges, Applied Physics Reviews **6**, 021314 (2019).

[48] T. P. Harty, D. T. C. Allcock, C. J. Ballance, *et al.*, High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit, Phys. Rev. Lett. **113**, 220501 (2014).

[49] C. J. Ballance, T. P. Harty, N. M. Linke, *et al.*, High-fidelity quantum logic gates using trapped-ion hyperfine qubits, Phys. Rev. Lett. **117**, 060504 (2016).

[50] A. Erhard, J. J. Wallman, L. Postler, *et al.*, Characterizing large-scale quantum computers via cycle benchmarking, Nature Communications **10** (2019).

[51] W. Paul, Electromagnetic traps for charged and neutral particles, Rev. Mod. Phys. **62**, 531 (1990).

[52] E. A. Martinez, T. Monz, D. Nigg, *et al.*, Compiling quantum algorithms for architectures with multi-qubit gates, New Journal of Physics **18**, 063029 (2016).

[53] J. P. Gaebler, A. M. Meier, T. R. Tan, *et al.*, Randomized benchmarking of multiqubit gates, Phys. Rev. Lett. **108**, 260503 (2012).

[54] C. Monroe, R. Raussendorf, A. Ruthven, *et al.*, Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects, Phys. Rev. A **89**, 022317 (2014).

[55] M. Ringbauer, M. Meth, L. Postler, *et al.*, A universal qudit quantum processor with trapped ions, Nature Physics **18**, 1053 (2022).

[56] M. A. Nielsen, A simple formula for the average gate fidelity of a quantum dynamical operation, Physics Letters A **303**, 249 (2002).

[57] M. Dalgaard, F. Motzoi, and J. Sherson, Predicting quantum dynamical cost landscapes with deep learning, Phys. Rev. A **105**, 012402 (2022).

[58] V. Strassen, Gaussian elimination is not optimal, Numerische Mathematik **13**, 354 (1969).

[59] N. Khaneja, T. Reiss, C. Kehlet, *et al.*, Optimal control of coupled spin dynamics: design of nmr pulse sequences by gradient ascent algorithms, Journal of Magnetic Resonance **172**, 296 (2005).

[60] M. Dalgaard, F. Motzoi, J. H. M. Jensen, and J. Sherson, Hessian-based optimization of constrained quantum control, Phys. Rev. A **102**, 042612 (2020).

[61] A. Morningstar, M. Hauru, J. Beall, *et al.*, Simulation of quantum many-body dynamics with tensor processing units: Floquet prethermalization, PRX Quantum **3** (2022).

[62] I. L. Chuang and M. A. Nielsen, Prescription for experimental determination of the dynamics of a quantum black box, Journal of Modern Optics **44**, 2455 (1997).

[63] V. Bužek and M. Hillery, Quantum copying: Beyond the no-cloning theorem, Phys. Rev. A **54**, 1844 (1996).

[64] S. Khatri, R. LaRose, A. Poremba, *et al.*, Quantum-assisted quantum compiling, Quantum **3**, 140 (2019).

[65] J. R. McClean, S. Boixo, V. N. Smelyanskiy, *et al.*, Barren plateaus in quantum neural network training landscapes, Nature Communications **9**, 4812 (2018).

[66] M. Cerezo, A. Sone, T. Volkoff, *et al.*, Cost function dependent barren plateaus in shallow parametrized quantum circuits, Nature Communications **12**, 1791 (2021).

[67] S. Jerbi, J. Gibbs, M. S. Rudolph, *et al.*, The power and limitations of learning quantum dynamics incoherently, preprint at arXiv:2303.12834 (2023).

[68] O. Kyriienko and V. E. Elfving, Generalized quantum circuit differentiation rules, Phys. Rev. A **104**, 052417 (2021).

[69] D. Wierichs, J. Izaac, C. Wang, and C. Y.-Y. Lin, General parameter-shift rules for quantum gradients, Quantum **6**, 677 (2022).

[70] L. Bittel, J. Watty, and M. Kliesch, Fast gradient estimation for variational quantum algorithms, preprint at arXiv:2210.06484 (2022).

[71] M. Ostaszewski, E. Grant, and M. Benedetti, Structure optimization for parameterized quantum circuits, Quantum **5**, 391 (2021).

[72] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, Reinforcement learning for combinatorial optimization: A survey, Computers & Operations Research **134**, 105400 (2021).

[73] Y. Li, Deep reinforcement learning: An overview, preprint at arXiv:1701.07274 (2017).

[74] W. L. Boyajian, J. Clausen, L. M. Trenkwalder, *et al.*, On the convergence of projective-simulation–based reinforcement learning in markov decision processes, Quantum Machine Intelligence **2**, 13 (2020).

[75] J. Mautner, A. Makmal, D. Manzano, *et al.*, Projective simulation for classical learning agents: A comprehensive investigation, New Generation Computing **33**, 69 (2015).

[76] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, Human-level control through deep reinforcement learning, Nature **518**, 529 (2015).

[77] S. Hochreiter, Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis (1991).

[78] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Computation **9**, 1735 (1997).

[79] D. P. DiVincenzo, Two-bit gates are universal for quantum computation, Phys. Rev. A **51**, 1015 (1995).

[80] M. Bukov, A. G. Day, D. Sels, *et al.*, Reinforcement learning in different phases of quantum control, Physical Review X **8** (2018).

[81] D. C. Liu and J. Nocedal, On the limited memory bfgs method for large scale optimization, Math. Program. **45**, 503 (1989).

[82] T. Toffoli, Reversible computing, in *Automata, Languages and Programming*, edited by J. de Bakker and J. van Leeuwen (Springer Berlin Heidelberg, Berlin, Heidelberg, 1980) pp. 632–644.

[83] T. Monz, K. Kim, W. Hänsel, *et al.*, Realization of the quantum toffoli gate with trapped ions, Phys. Rev. Lett. **102**, 040501 (2009).

[84] A. G. Taube and R. J. Bartlett, New perspectives on unitary coupled-cluster theory, International Journal of Quantum Chemistry **106**, 3393 (2006).

[85] R. J. Bartlett and M. Musiał, Coupled-cluster theory in quantum chemistry, Rev. Mod. Phys. **79**, 291 (2007).

[86] F. Motzoi, M. P. Kaicher, and F. K. Wilhelm, Linear and logarithmic time compositions of quantum many-body operators, Phys. Rev. Lett. **119**, 160503 (2017).

[87] M. Takahashi, *Thermodynamics of One-Dimensional Solvable Models* (Cambridge University Press, 1999).

[88] S. K. Lam, A. Pitrou, and S. Seibert, Numba: A llvm-based python jit compiler, in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (2015) pp. 1–6.

[89] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, JAX: composable transformations of Python+NumPy programs (2018).

[90] A. Paszke, S. Gross, F. Massa, *et al.*, Pytorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, *et al.* (Curran Associates, Inc., 2019) pp. 8024–8035.

[91] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods **17**, 261 (2020).

[92] J. Li, X. Yang, X. Peng, and C.-P. Sun, Hybrid quantum-classical approach to quantum optimal control, Phys. Rev. Lett. **118**, 150503 (2017).

[93] N. Barhate, Minimal pytorch implementation of proximal policy optimization, code: `https://github.com/nikhilbarhate99/PPO-PyTorch` (2021).

[94] P. E. Hart, N. J. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics **4**, 100 (1968).

[95] A. Hodler and M. Needham, *Graph algorithms* (O'Reilly Media, Sebastopol, CA, 2019).

A Fast analytic ion gates

In this section, we derive the representations for the MS and C_{xy} gates based on their spectral decomposition and we show how these can be further simplified. We also provide descriptions of the gate gradients based on the optimized representations. The n-qubit XY-Rotation and MS Hamiltonians are given by

$$\hat{H}_{\mathrm{XY}}(n,\phi) = S_x\cos(\phi) + S_y\sin(\phi), \tag{33}$$

$$\hat{H}_{\mathrm{MS}}(n,\phi) = \hat{H}_{\mathrm{XY}}(n,\phi)^2 = \left(S_x\cos(\phi) + S_y\sin(\phi)\right)^2, \tag{34}$$

We will first focus on solving the XY-rotation gate and then generalize our solution to the MS gate, using the fact that the MS-Hamiltonian is the square of the XY-Hamiltonian. A representation of the XY unitary in terms of its real and imaginary entries for $n = 8$, $\theta = \frac{3}{4}\pi$ and $\phi = 2\pi$ is given in Fig. 6.

A.1 General Approach - Constructing the XY-rotation Gate

Due to the lack of multi-qubit interaction terms in the XY-rotation Hamiltonian, the eigenvectors and eigenvalues of the n-qubit case can be constructed from the single qubit closed-form solution, allowing us to factorize the evolution into ϕ dependent terms and θ dependent terms (θ can be understood as the time evolution parameter, using a standard notation of the literature of trapped-ion quantum computing). This allows us to calculate the associated unitaries from element-wise operations. We decompose the Hamiltonian into

$$\hat{H}_{\mathrm{XY}}(n,\phi) = \hat{V}_n(\phi)\Lambda_{\mathrm{XY}}\hat{V}_n^\dagger(\phi), \tag{35}$$

with the ϕ dependent $\hat{V}_n$ and the diagonal matrix of the eigenvalues Λ_{XY}. For the unitaries the eigenvalues then capture the θ dependence via $\exp\left(-i\Lambda_{\mathrm{XY}}\frac{\theta}{2}\right)$.

Eigenvalues The eigenvalues are constructed from the single qubit case using the following Kronecker sum notation $\hat{A}^{\oplus n} = \sum_{i=1}^{n}\mathbb{I}^{\otimes i-1}\otimes\hat{A}\otimes\mathbb{I}^{\otimes n-i}$,

$$\Lambda_{\mathrm{XY}} = \mathrm{diag}\left(2\mathbf{b}_n - n\right), \quad \text{with} \quad \mathbf{b}_n = \begin{pmatrix}0\\1\end{pmatrix}^{\oplus n}. \tag{36}$$

We find that the i-th eigenvalue can be constructed from the binary Hamming weight vector $\mathbf{b}_n$, because the i-th component of the binary Hamming weight $b_n(i)$ corresponds to the number of qubits with $|1\rangle$ at index i of the Hilbert space.

Eigenvectors The eigenvector matrices $\hat{V}_n(\phi)$ are also constructed from the single qubit case, but using the tensorproduct,

$$\hat{V}_n(\phi) = \frac{1}{\sqrt{2^n}}\begin{pmatrix}1 & 1\\ -e^{i\phi} & e^{i\phi}\end{pmatrix}^{\otimes n} = \hat{P}_n(\phi)\odot\frac{1}{\sqrt{2^n}}\underbrace{(\hat{\mathbb{I}}_2 + i\sigma_y)^{\otimes n}}_{=\hat{S}_n}. \tag{37}$$

We furthermore decompose them into an elementwise product ($\odot$) of a sign $\hat{S}_n$ and phase $\hat{P}_n(\phi)$ component. The phase components $\hat{P}_n(\phi)$ are column-independent $\hat{P}_n(\phi) = [\mathbf{p}_n(\phi), \mathbf{p}_n(\phi), \cdots, \mathbf{p}_n(\phi)]$, where the vectors themselves can be regarded as the element wise complex exponential v_ϕ of the binary Hamming weight vector $\mathbf{b}_n$

$$\mathbf{p}_n(\phi) = \begin{pmatrix}1\\ e^{i\phi}\end{pmatrix}^{\otimes n} = \exp(i\phi\mathbf{b}_n) = v_\phi(\mathbf{b}_n). \tag{38}$$

Calculating the Unitary We can now use the column independence of the phase components $\hat{P}_n(\phi)$, to commute it with the eigenvalues. This allows us to separate the phase component further. For the unitary of the XY-gate we then find

$$\begin{aligned} C_{xy}(\phi,\theta) &= \exp\left(-i\hat{H}_{\mathrm{XY}}(\phi)\theta\right) = \hat{V}_n(\phi)\exp(-i\Lambda_{\mathrm{XY}}\theta)\hat{V}_n^\dagger(\phi)\\ &= \frac{1}{2^n}\underbrace{\hat{P}_n(\phi)\hat{P}_n^\dagger(\phi)}_{=v_\phi(\hat{C})}\hat{S}_n\exp(-{}^{i\theta}/_{2}\Lambda_{\mathrm{XY}})\hat{S}_n^\dagger. \end{aligned} \tag{39}$$

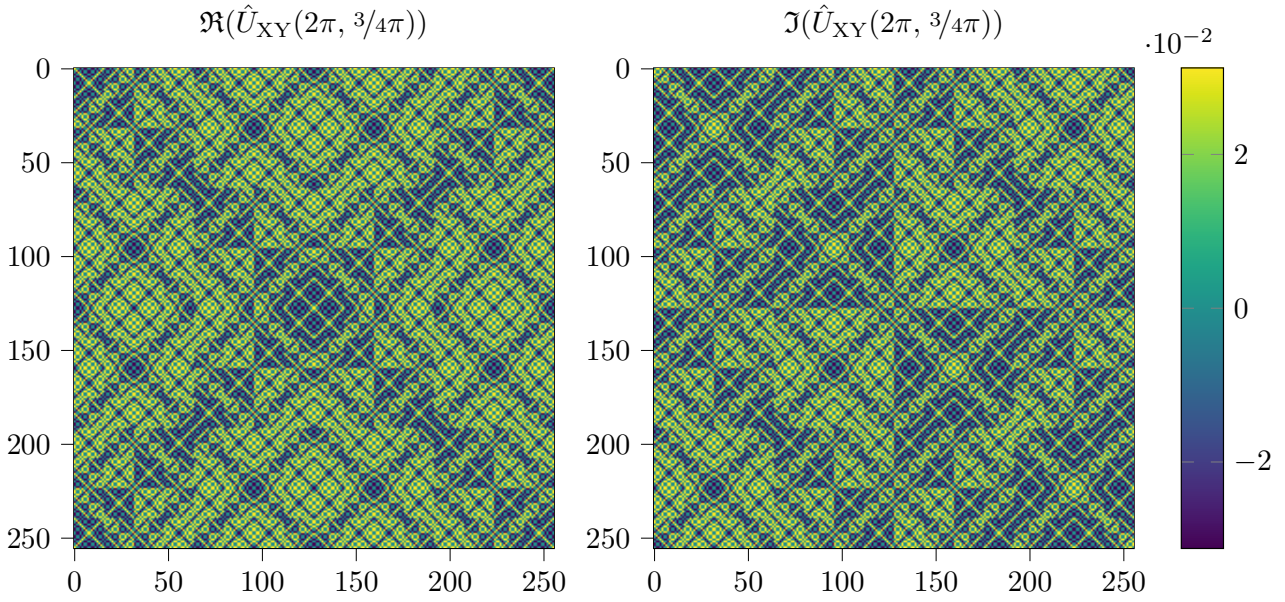


Figure 6: Real and imaginary components of the unitary matrix of the n-qubit XY-rotation gate with $\phi = 2\pi$ and $\theta = {}^3\!/_4\pi$. The colormap describes the value range of the matrix elements.

We substitute the phase term product with the previously defined element wise complex exponential of $\hat{C} = \hat{P}_n - \hat{P}_n^T$. Furthermore we split the sign components into degeneracy subspaces one for each different eigenvalue in λ_{XY}. We decompose $\hat{S}_n = \sum_{i=0}^{n} \hat{S}_{\lambda_i}$, where $\hat{S}_{\lambda_i}$ retains the values of $\hat{S}_n$ for columns with binary Hamming weight i, but is zero for all other columns. From this we can then construct cached matrices $\hat{D}_{\lambda_i} = \hat{S}_{\lambda_i}\hat{S}_{\lambda_i}^\dagger$, so that we can rewrite

$$\hat{S}_n \exp(-i\theta/2\Lambda_{\mathrm{XY}})\hat{S}_n^\dagger = \sum_{i=0}^{n} e^{-i\lambda_{\mathrm{XY},i}\frac{\theta}{2}}\hat{D}_{\lambda_i}. \tag{40}$$

From this we conclude

$$C_{xy}(\phi,\theta) = \frac{1}{2^n} v_\phi(\hat{C}) \odot \sum_{i=0}^{n} \exp(-i\lambda_{\mathrm{XY,i}}\theta)\ \hat{D}_{\lambda_i}. \tag{41}$$

The element wise exponential $v_\phi(\hat{C})$ is constructed efficiently by reusing $2n+1$ phase values, because $C_{ij} \in [0, 1, \cdots, 2n]$. Each of the required computations require $\mathcal{O}\left((2^n)^2\right)$ operations, resulting in a total complexity of $\mathcal{O}\left((n+1)(2^n)^2\right)$.

A.2 Generalization to the MS gate

The approach used for C_{xy} gate can be generalized to the MS gate. Due to its Hamiltonian being the square of the aforementioned Hamiltonian, the two gates share their eigenvectors, while the eigenvalues of the C_{xy} gate are squared $\lambda_{\mathrm{XY,i}}^2 = \lambda_{\mathrm{MS,\,i}}$. This reduces the number of different eigenvalues from $n+1$ to $\lceil \frac{n}{2} \rceil +\ \mod (2+1, 2)$. The expansion in Eq. (41) is transformed into

$$\mathrm{MS}(\phi,\theta) = \ \tfrac{1}{2^n} v_\phi(\hat{C}) \odot \left(\sum_{i=0}^{\lceil \frac{n}{2} \rceil} \exp(-i\lambda_{\mathrm{MS,i}}\theta)\ \overbrace{\left(\hat{D}_{\lambda_i} + \hat{D}_{\lambda_{n-i}}\right)}^{=\tilde{D}_i} \delta_i^{\mathrm{floor}} \right). \tag{42}$$

where

$$\delta_i^{\mathrm{floor}} = \begin{cases} 1 & \text{if } 0 \le i \le \frac{n}{2} \\ 0 & \text{otherwise.} \end{cases} \tag{43}$$

As both expansions, Eq. (41) and Eq. (42), rely solely on operations local to the matrix elements, these operations are ideal for parallelisation. In Fig. 7 we show the improvement in terms of computation time (a) and speedup

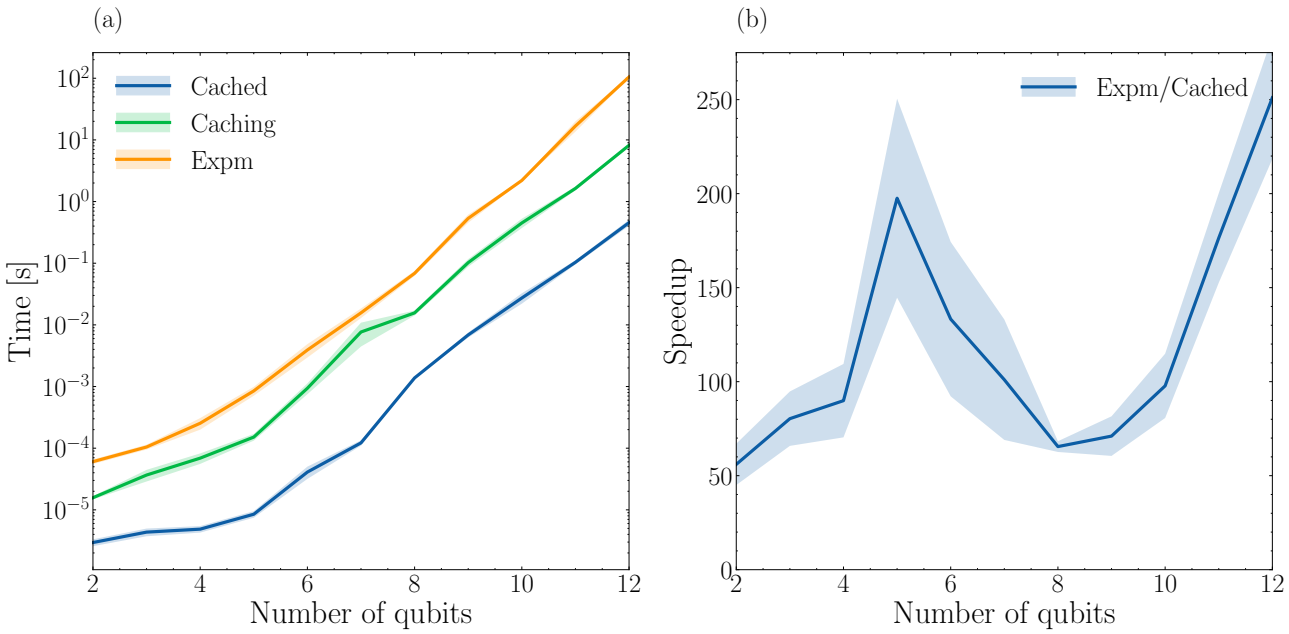


Figure 7: (a) Walltime for MS gate construction with standard matrix exponentiation and with the analytical approach of this paper as a function of the number of qubits. The orange line corresponds to the function *scipy.expm*, the blue and green lines represent the analytical approach during caching and after caching, respectively; (b) speedup of the new approach, once the parameter-indedependent matrices have been cached, as a function of the number of qubits. Here we see that the speedup is not constant as the number of qubits changes, but we have a maximum speedup around 5 qubits, which then decreases until we reach 8 qubits, and then increases again. This effect is most likely due to an underlying slowdown in the underlying fundamental operations, e.g., due to the processor cache being filled up quickly, so that access to the RAM becomes necessary. Moreover, our method is implemented only using NUMBA, i.e., compiled PYTHON code, whereas the exponential matrix function of SCIPY profits from underlying time-critical routines written in C, C++ and Fortran. It seems, however, that the speedup grows steadily for values larger than 8 qubits.

(b) of our method compared to standard matrix exponentiation as a function of the number of qubits : in (a) the orange line shows the computation time of the matrix exponential, whereas the green line shows our method implemented without caching the parameter-independent matrices, and the blue line our method again, but with cached matrices; (b) shows instead the ratio between the computation time of the cached method and standard matrix exponentiation. We see that we can reach a speedup between 50 and 250 for $n \leq 12$, which proves particularly useful in our quantum-circuit simulations: In fact, these require large numbers of cost function evaluations, due to the presence of both the reinforcement learning agent and the gradient-based optimization.

A.3 Gradients and Hessians

The gradient of the C_{xy} gate can then be computed analytically via

$$\frac{\partial}{\partial\phi}C_{xy}(\phi,\theta) = \quad i\hat{C}\tfrac{1}{2^n}v_\phi(\hat{C}) \odot \textstyle\sum_{i=0}^{n} \exp(-i\lambda_{\mathrm{XY,i}}\theta)\ \hat{D}_{\lambda_i}, \tag{44}$$

$$\frac{\partial}{\partial\theta}C_{xy}(\phi,\theta) = \quad \tfrac{1}{2^n}v_\phi(\hat{C}) \odot \textstyle\sum_{i=0}^{n} -i\lambda_{\mathrm{XY,i}} \exp(-i\lambda_{\mathrm{XY,i}}\theta)\ \hat{D}_{\lambda_i}. \tag{45}$$

For the derivative by ϕ the sum can be reused from the previous gate construction in Eq. (41) further reducing computational demands. For the MS gate we find

$$\frac{\partial}{\partial\phi}\mathrm{MS}(\phi,\theta) = \quad i\hat{C}\tfrac{1}{2^n}v_\phi(\hat{C}) \odot \left(\textstyle\sum_{i=0}^{\lceil \frac{n}{2} \rceil} \exp(-i\lambda_{\mathrm{MS,i}}\theta)\ \overbrace{\left(\hat{D}_{\lambda_i} + \hat{D}_{\lambda_{n-i}}\right)}^{=\tilde{D}_i} \delta_i^{\mathrm{floor}} \right) \tag{46}$$

$$\frac{\partial}{\partial\theta}\mathrm{MS}(\phi,\theta) = \quad \tfrac{1}{2^n}v_\phi(\hat{C}) \odot \left(\textstyle\sum_{i=0}^{\lceil \frac{n}{2} \rceil} -i\lambda_{\mathrm{MS,i}} \exp(-i\lambda_{\mathrm{MS,i}}\theta)\ \overbrace{\left(\hat{D}_{\lambda_i} + \hat{D}_{\lambda_{n-i}}\right)}^{=\tilde{D}_i} \delta_i^{\mathrm{floor}} \right). \tag{47}$$

Hessians can be derived direcly by applying the chain rule a second time in Eqs. (44)-(47).

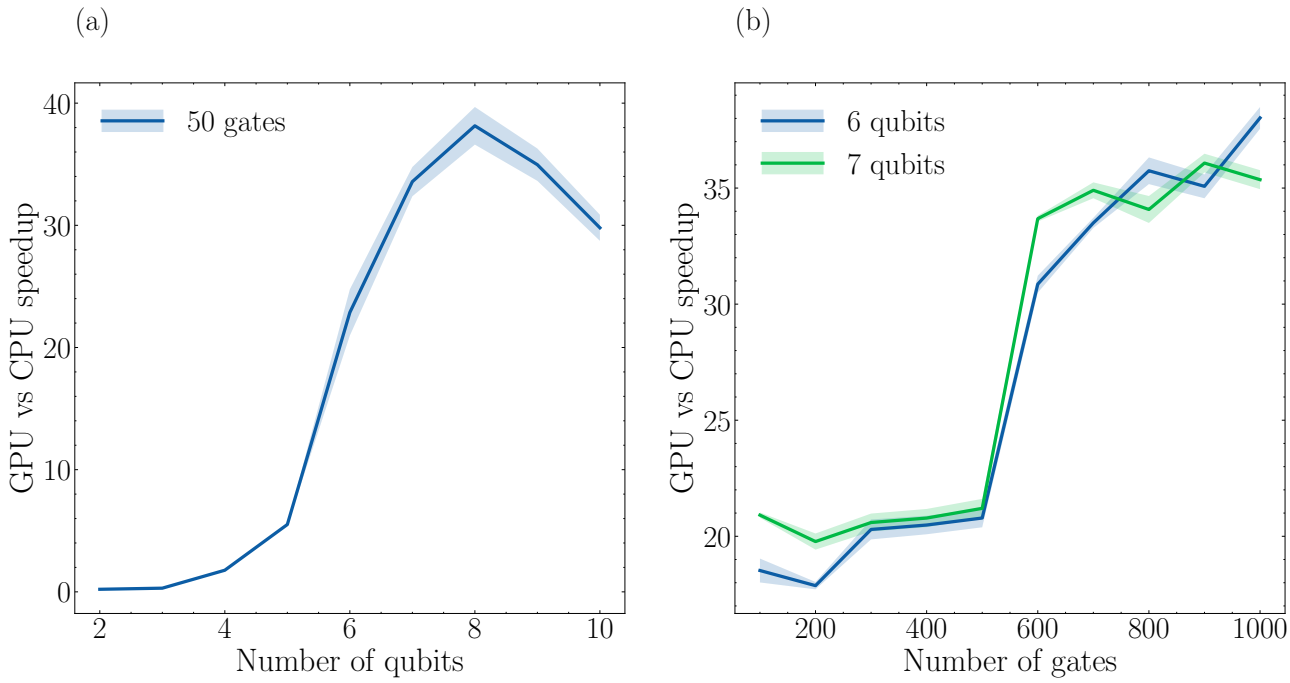


Figure 8: Average speedup (30 shots) required to compute the gradient for: (a) the same circuit (50 gates) with increasing numbers of qubits. (b) a 6-qubit (blue) and 7-qubit (green) gate set for increasing number of random gates on the circuit. Shaded regions show the standard deviation for each point (vertical bars are connected together to form a poligon). The orange line represents the gradient compiled with NUMBA on CPU (Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz), the second one the gradient compiled with JAX on GPU (NVIDIA Tesla V100S-PCIE-32GB). We see a slowdown in the speedup as we move past 8 qubits: this is probably due to computational overheads that do not profit from the GPU parallel calculations. In fact, we see that the slowdown is present only as we increase the Hilbert space dimension and not as we increase the number of gates.

B Parameter-shift rules for ion gates

The method for gate computation introduced above allows to express the derivative of a quantum cost function in terms of so-called parameter-shifts rules. These have been studied in the context of variational quantum circuit, quantum control and quantum machine learning. The gates contained in the gate set $Z_1(\theta), ..., Z_n(\theta)$, $C_{xy}(\theta, \phi)$, $\mathrm{MS}(\theta, \phi)$. In general, expectation values with respect to parametrized quantum circuits will have the form

$$C(\boldsymbol{\alpha}) = \langle\psi| V(\boldsymbol{\alpha}) \rho V^\dagger(\boldsymbol{\alpha}) |\psi\rangle , \tag{48}$$

for a given quantum state $|\psi\rangle$, where $V(\boldsymbol{\alpha})$ is a gate in the gateset defined in Section (2). The cost function depends on a specific target operator ρ.

We first show that the cost function in Eq. (48) is equivalent to the cost function of Eq. (13), as well as to the expectation value of the local and non-local Hilbert Schmidt circuit, which is given in Ref. [64]:

$$C_{HST}(\boldsymbol{\alpha}) = 1 - \mathrm{Tr}\big\{H(U \otimes V(\boldsymbol{\alpha})^*)\rho(U^\dagger \otimes V(\boldsymbol{\alpha})^T)\big\}, \tag{49}$$

with $\rho = H = |\phi_+\rangle_{A,B} \langle\phi_+|_{A,B}$ for the Hilbert Schmidt test, where A and B are the subsystems for unitary U and V and

$$|\phi_+\rangle_{A,B} = \frac{1}{\sqrt{d}} \sum_{\boldsymbol{j}} |\boldsymbol{j}_A\rangle \otimes |\boldsymbol{j}_B\rangle . \tag{50}$$

Proof. We show the equivalence for the Hilbert Schmidt test. The proof for the local Hilbert Schmidt test is analogous.
We see that

$$\mathrm{Tr}\big\{H(UV^\dagger \otimes I)\rho(V \otimes U^*)\big\} = \frac{1}{d} \sum_{\boldsymbol{j}} \mathrm{Tr}\big\{|\boldsymbol{j}_A\rangle \langle\boldsymbol{j}_A| UV^\dagger |\boldsymbol{j}_A\rangle \langle\boldsymbol{j}_A| VU^\dagger\big\} \mathrm{Tr}\{|\boldsymbol{j}_B\rangle \otimes \langle\boldsymbol{j}_B|\},$$

which can be represented as a sum of squared amplitudes of with the same structure of (48). □

As a result, the cost functions considered here have to a common description, similar to Eq. (48). We derive and comment the corresponding parameter-shift rule for each one of these gates. We consider here the shift of one single parameter at a time, i.e., a scalar rotation angle θ. The gradient is constructed by shifting each parameter according to its own specific parameter-shift rule. The derivatives of the function can be written as:

$$\frac{\partial}{\partial\theta}C(\theta) = \langle\psi| \frac{\partial}{\partial\theta}V(\theta)UV^\dagger(\theta) + V(\theta)U\frac{\partial}{\partial\theta}V^\dagger(\theta)\,|\psi\rangle\,. \tag{51}$$

Assuming the gate has a generator $V(\theta) = e^{iG\theta}$, where G is a parameter-independent hermitian matrix, then we have:

$$\frac{\partial}{\partial\theta}C(\theta) = \langle\psi|\, V(\theta)i[G,U]V^\dagger(\theta)\,|\psi\rangle\,. \tag{52}$$

In the simplest case, we consider, e.g., $G = \sigma_z^{(i)}$ and obtain [92]:

$$[\sigma_z^{(i)}, U] = e^{i\sigma_z^{(i)}\pi/2}Ue^{-i\sigma_z^{(i)}\pi/2} - e^{-i\sigma_z^{(i)}\pi/2}Ue^{i\sigma_z^{(i)}\pi/2}, \tag{53}$$

which leads to

$$\frac{\partial}{\partial\theta}C(\theta) = C\left(\theta + \frac{\pi}{2}\right) - C\left(\theta - \frac{\pi}{2}\right), \tag{54}$$

a parameter-shift rule which is valid for the local Z-rotations. For the other two gates, we have to differentiate with respect to both the parameters θ and ϕ. For θ, applying Eq. (15) in Ref. [69], we have:

$$\frac{\partial}{\partial\theta}C(\theta,\phi)\Big|_{\theta=0} = \sum_{l=1}^{2n}\frac{(-1)^{l-1}}{2\sin\left(\frac{2l-1}{2n}\pi\right)}C\left(\frac{2l-1}{2n}\pi,\phi\right). \tag{55}$$

By using the general decomposition of the C_{xy} gates, parameter-shift rules can be obtained directly by studying how the eigenvalues and eigenvectors of the unitaries vary as a function of the continuous parameters. Since the C_{xy} gate has $n+1$ distinct eigenvalues we will need at most $2(n+1)$ samples to calculate the derivative – see [69]. We observe that the matrices $\hat{V}_n(\phi)$ in Eq. (37) can be decomposed further in elementary gates, such as:

$$\hat{V}_n(\phi) = (\mathcal{P}(\phi)HX)^{\otimes n} = \mathcal{P}(\phi)^{\otimes n}H^{\otimes n}X^{\otimes n}, \tag{56}$$

where

$$\mathcal{P}(\phi) = \begin{pmatrix}1 & 0\\ 0 & e^{i\phi}\end{pmatrix}, H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, X = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}. \tag{57}$$

Let us first consider the C_{xy} gate. The corresponding Hamiltonian has $n+1$ different eigenvalues with energy $\lambda_i = 2i - n$ and $c_k(\phi) = v_k(\phi)^\dagger U v_k(\phi)$ are the overlaps between the target operator U and the gate eigenvectors $v_k(\phi)$ of the gate. For the set of difference pairs we have $\lambda_i - \epsilon_j = 2(i-j) := 2l$. Then Eq. (55) becomes

$$\frac{\partial}{\partial\theta}C(\theta,\phi)\Big|_{\theta=0} = \sum_{k=1}^{n}\sum_{l=1}^{2n}\frac{(-1)^{l-1}}{2\sin\left(\frac{2l-1}{2n}\pi\right)}c_k(\phi)e^{2ik\left(\frac{2l-1}{2n}\right)}. \tag{58}$$

.

This expression can be simplified by evaluating the sum at $\theta = 0$ with respect to the index l.

$$\frac{\partial}{\partial\theta}C(\theta,\phi)\Big|_{\theta=0} = \sum_{k=1}^{n}c_k(\phi)\frac{\csc\left(\frac{\pi}{2n}\right)e^{\frac{\pi i(k(4n+2)-n)}{n}}\left(e^{\frac{2\pi i(n+1)(n-2k)}{n}} - (-1)^{n+\frac{1}{n}}\right)}{2n\left((-1)^n e^{4\pi ik} + e^{2\pi in}\right)}. \tag{59}$$

For the MS gate we obtain instead the following expression:

$$\frac{\partial}{\partial\theta}C(\theta,\phi)\Big|_{\theta=0} = \sum_{k=1}^{n}c_k(\phi)\left\{-\frac{\csc\left(\frac{\pi}{2n}\right)e^{8\pi ikn-\frac{\pi i(n-2k)^2}{n}}\left((-1)^{n+\frac{1}{n}}e^{\frac{2\pi i(n+1)(n-2k)^2}{n}} - 1\right)}{2n\left((-1)^n e^{2\pi i(4k^2+n^2)} + e^{8\pi ikn}\right)}\right\}. \tag{60}$$

For the derivative of Eq. (48) with respect to ϕ, we just have to consider the gate $\mathcal{P}(\phi)$ in Eq. (56), whose single-qubit gate generator is given in Eq. (61). We observe that due to the simple structure of the gate, the derivative of the $C_{x,y}$ gate with respect to ϕ is simply given by:

$$\mathcal{P}(\phi) = \exp\left\{\frac{i\phi}{2}(I_n - S_z)\right\} = \exp\left\{\frac{i\phi}{2}\right\}\exp\left\{-\frac{i\phi}{2}S_z\right\}, \tag{61}$$

where $S_z = \sum_{i=1}^{n}\sigma_z^{(i)}$ is the collective Z-operator acting on the entire qubit register. After inserting Eq. (61) into Eq. (48) for a C_{xy} gate, and using the representation in Eq. (35), we obtain

$$C(\theta,\phi) = \langle\psi|\left(\exp\left\{-\frac{i\phi}{2}S_z\right\}B_{\mathrm{XY}}(\theta)\exp\left\{\frac{i\phi}{2}S_z\right\}U\exp\left\{-\frac{i\phi}{2}S_z\right\}B_{\mathrm{XY}}(\theta)^\dagger\exp\left\{\frac{i\phi}{2}S_z\right\}\right)|\psi\rangle\,, \tag{62}$$

where $B_{\mathrm{XY}}(\theta) = H^{\otimes n}X^{\otimes n}\left(\sum_{l=0}^{n}\exp\{-i\lambda_l\theta\}\,|l\rangle\langle l|\right)X^{\otimes n}H^{\otimes n}$ – the same equation holds for the MS gate, one needs just to replace λ_{XY} with the corresponding eigenvalues λ_{MS}. Considering that $\mathcal{P}(\phi)$ has two eigenvalues, we can write

$$\exp\left\{-\frac{i\phi}{2}S_z\right\} = e^{i\phi}\Pi_z^{(1)} + e^{-i\phi}\Pi_z^{(2)}, \tag{63}$$

where $\Pi_z^{(1)}$ and $\Pi_z^{(2)}$ are the projectors on the respective degenerate eigenspaces. By using Eq. (63) in Eq. (62), we see that Eq. (62) has the following form

$$C(\theta,\phi) = b_0(\theta) + b_1(\theta)e^{-i\phi} + b_2(\theta)e^{i\phi} + b_3(\theta)e^{2i\phi} + b_4(\theta)e^{-2i\phi}, \tag{64}$$

where $b_i(\theta), i = 0,1,2,3,4$ are ϕ-independent coefficients. Thus, we can write the derivative of C with respect to ϕ as

$$\frac{\partial}{\partial\phi}C(\theta,\phi) = -ib_1(\theta)e^{-i\phi} + ib_2(\theta)e^{i\phi} + b_3(\theta)2ie^{2i\phi} - 2ib_4(\theta)e^{-2i\phi}, \tag{65}$$

which can be written as a linear combination of $C(\theta,\phi\pm\frac{\pi}{2})$ and $C(\theta,\phi\pm\frac{\pi}{4})$ cost functions:

$$\frac{\partial}{\partial\phi}C(\theta,\phi) = \tilde{C}(\theta,\phi+\frac{\pi}{4}) - \tilde{C}(\theta,\phi-\frac{\pi}{4}) + (\frac{1}{\sqrt{2}}-\frac{1}{2})\tilde{C}(\theta,\phi-\frac{\pi}{2}) + (\frac{1}{\sqrt{2}}+\frac{1}{2})\tilde{C}(\theta,\phi+\frac{\pi}{2}). \tag{66}$$

C Algorithmic implementation details

Our framework consists of two main parts: The agent (we consider a PS-LSTM agent with LSTM cells and a linear layer, but any RL agent is a viable option), which should learn to construct a proper representation of the quantum circuit and the optimizer, which has to be equipped with a proper gradient function. The gradient function is constructed according to the standard GRAPE procedure and using the gate representations for the C_{xy} and the MS gates given in Appendix A. We use and test two different versions of the gradient: The first one is compiled using NUMBA [88], a library for fast python code, the second one employs JAX to allow execution on GPU. A comparison of the two gradient functions is given in Fig. 8. We observe that the GPU-based function allows for a certain speed-up, in particular as the number of gates on the circuit increases. The main advantage of NUMBA lies in a faster and more straightforward implementation of the parallel optimization runs with different seeds.

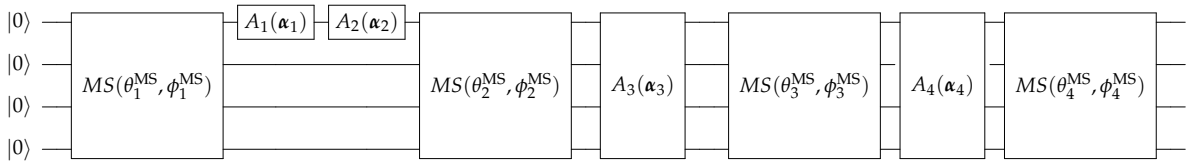
At time t of the agent-environment interaction, the agent receives an input (percept) from the environment and outputs an action which is executed onto the environment. The agent-environment interaction has the following structure.

Action: The action of the agent corresponds to placing one of the available gates in the gate set (entangling or non-entangling) onto the quantum circuit. The gate is represented by an integer $a \in 1,...,|A|$. The array of all the integers chosen by the agent up to time t forms the quantum circuit structure at time t.

Percept: As input to the RL agent, we generally use a one-hot encoding of the circuit. For a circuit of length L and $|A|$ different gates, the percept $s_t \in \{0,1\}^L \times \{0,1\}^{|A|}$ has entries equal to:

$$(s_t)_{i,j} = \begin{cases} 1 & \text{if } i,j = a,t \\ 0 & \text{otherwise.} \end{cases} \tag{67}$$

(a) Hybrid layer-based-RL environment with action space $A = \{C_{xy}, Z_1, ..., Z_n\}$ and fixed MS gates.



(b) Full RL environment with action space $A = \{\text{MS}, C_{xy}, Z_1, ..., Z_n\}$.

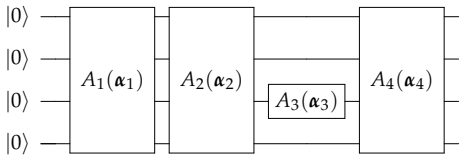


Figure 9: Quantum circuit representation of two possible RL environments for quantum circuit optimization. The first environment (a) resembles the structure of the layer-based compilation, that is, the entangling MS gates are fixed and the agent can place rotations on the circuit between the entangling layers. The second one (b) has no pre-defined circuit structure but rather leaves the agent completely free to place any gate withing the gate set on the quantum circuit, with only one additional simplification: two gates of the same type placed immediately one after the other are automatically merged to form one single gate. This is done to prevent the agent from getting stuck in loops, i.e., local minima, where it keeps choosing the same gate over and over again. In general, the first circuit reduces the size of the action space and therefore the possible shapes of the corresponding cost function, hence reducing exploration in favour of a more standardized search.

However, for the PS-LSTM agent (or other agents that are modified analogously), a more suitable input can also be used. Since the update of the internal state of the PS-LSTM agent follows the RL steps, the agent can just accept the input at time t as a percept, since previous information about past agent-environment interactions is still processed by the the internal state of the LSTM network. Therefore, the percept becomes the one-hot encoded vector

$$(s_t)_j = \begin{cases} 1 & \text{if } j = a \\ 0 & \text{otherwise.} \end{cases} \tag{68}$$

This percept only gives the agent partial information about the internal state of the environment, which turns the problem from an MDP (Markov Decision Process) into a so-called POMDP problem (Partially Observable Markov Decision Process) [42]. Other agents than those derived by PS may use different inputs, based on their specific convergence properties in dealing with different environments. In general, if the agent can accept a recurrent or autoregressive network as a policy, the percept given in Eq. (68) may be used. It could be, however, that the algorithm needs to be modified appropriately to function with this type of percept. The first type of input has been used for the simulation of UCC operators, whereas the second one has been implemented in all the other simulations. In general, both percepts lead to similar results, but the second one should be preferred for the PS-LSTM architecture and most importantly does not scale with the size of the circuit, leading therefore to faster forward passages in the policy network.

Reward and curriculum: Throughout the development of this manuscript, several reward systems were tested. In general, we used a two-step reward that assigns a smaller reward value when the cost function minimum falls below the actual curriculum threshold ϵ_t and a larger reward value when the cost function minimum falls below the global target threshold $\epsilon_{\min}$. Both curriculum thresholds can be adjusted depending on the gate synthesis problem to be tackled. The episode terminates when the cost function minimum in a given time step falls below the threshold ϵ_t or when the maximal length of the circuit per episode, $L_{\max}$, is reached. The RL training terminates upon reaching the maximum number of episodes $E_{\max}$. The reward scheme helps to progressively increase the fidelity throughout training without allowing for too long circuits. The threshold is then lowered as episodes progress based on previous rewards obtained by the agent. In our implementation, we lower the threshold when it has been surpassed by the agent at least 500 times using the

following scheme:

$$\epsilon_{t+1} = \begin{cases} \epsilon_{\min} + \frac{1}{2}(\epsilon_{\min} - \epsilon_t) & \text{if } \epsilon_{\min} \leq \epsilon_t \leq 1 \\ \epsilon_{\min} & \text{if } \epsilon_t \leq \epsilon_{\min} \\ 1 & \text{otherwise} \end{cases} \tag{69}$$

Environments: We propose two different types of agent-circuit interaction. In the first type – see Fig. 9 (a) –, which is more similar to layer-based compilation, the agent only places rotations on the circuit, while the entangling gates are fixed in position. While the episode progresses, more entangling gates are added to the environment, until a maximum $L_{\max}$ of gates on the circuit is reached. The circuit is also simplified automatically by the environment, in such a way that if the agent places the same two gates on the circuit one after the other, they are reduced to one single gate. This is implemented in order to prevent the agent from getting trapped in local minima where it places the same gate over and over again on the circuit, therefore reducing the necessary training time. In the second type – see Fig. 9 (b) –, entangling gates are also given as a possible action on the environment. This second type of environment leaves more room for exploration, but it is also more challenging for the agent. The simulations presented in this work were realized by using only the environment that allows for completely free gate placement (so both the rotation gates as well as the entangling gates).

Agents: In order to test the effectiveness of our algorithm, we employ different agents for testing. We consider the standard state-of-the-art PPO algorithm [93] and the two versions of PS with deep energy-based neural networks PS-DEBN and PS-LSTM. We observe that PPO performs slightly worse than PS-LSTM, but better than PS-FNN, but is probably due to the non-recurrent version of the PPO algorithm implemented. Due to the large action and percept space, we did not include standard PS in the comparisons, since this would require to store large h-matrices for each percept-action transition, leading to slow computation and eventually memory overflow. This algorithm can however be used in circuit synthesis problems with a small number of gates and qubits.

Optimizers: As an unconstrained optimizer, in this work we only consider the L-BFGS-B algorithm, as it is implemented in SCIPY [91]. The number of iterations of the optimizers for each RL interaction is set to 100. Both optimizers can run a given number of optimization attempts in parallel with different seeds (a technique usually referred to as random restart), which should prevent the optimizer from getting trapped in local minima.

Simulations: In this work we consider three different groups of simulations: the synthesis of a 3-qubit Toffoli gate on a 3-qubit and 4-qubit circuit, the synthesis of UCC (Unitary Coupled Cluster) operators and the synthesis of the unitary of the XXZ Hamiltonian with varying Hamiltonian parameters. In the case of the XXZ Hamiltonian and the Toffoli gate, the simulation is realized on CPUs (72 Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz), whereas the simulation of the UCC operators is performed on GPUs (4 NVIDIA A100 Tensor Core GPU with 40 GB). This means, the library used to compute cost functions and gradients in order to optimize them is NUMBA for the CPU-based simulations and JAX for GPU-based ones. The reason is that NUMBA allows for faster parallelization with the JOBLIB library that proves difficult to achieve with JAX, thereby enabling us a better exploration of the cost function landscape for small numbers of qubits, whereas JAX is significantly faster for larger numbers of qubits. In particular, we set the number of optimization runs per RL iteration to 10 when we employ the NUMBA version, whereas we use only 1 when we employ the code using JAX. The curriculum threshold was kept to $\epsilon_{\min} = 10^{-2}$. We used two-layered LSTM networks implementing the policy given in Eq. (23) and 128 neurons, a training batch size of 64, a learning rate of 0.01, a curriculum update window of 500. The agent is trained with a replay-memory upon finding a gate sequence with infidelity lower than the curriculum threshold, and also every 100 agent-environment interactions if the replay memory is large enough. The target network is updated every 50 agent-environment interactions. The number of iterations of the optimizer in the hybrid RL-continuous simulations is fixed to 100. The temperature parameter β of the softmax distibution that parametrized the PS-LSTM policy – see Eq. (19) – is annealed from $\beta = 10^{-3}$ to $\beta = 1$ according to a linear schedule based on the number of episodes, in order to force the agent, towards the end of the training, to consider shorter and shorter circuits, thereby reducing the exploration. Other parameters vary based on the simulation considered. The data and hyperparameters can be found in https://github.com/franz3105/RL_Ion_gates.

Comparison with BQSKit: BQSKit is a quantum circuit compilation library developed by the Berkeley

Gate type	A^* search	Dijkstra	Greedy	our method
MS gate	5	3	4	3
$C_{x,y}$ gate	3	4	94	4
Z gate	27	24	297	3
Depth	17	15	113	10
Runtime	17 s	63 s	161 s	1.6 h

Table 1: Result of the compilation of the 3-qubit Toffoli gate with BQSKit [7] using the gate set given by Eqs. (1)-(3) and three of the given search heuristics: A^*, Dijkstra, and greedy. The threshold is set to $\epsilon = 10^{-2}$. We see that the compilation uses a considerable amount of Z gates and a larger amount of collective gates than the RL algorithm. We see, however, that a standard compiler can be significantly faster than a RL-based search with 10000 episodes and it is therefore possible to apply further pruning methods and iterative optimization loops on top of the main compilation algorithm.

National Laboratory [7]. It supports compilation of both discrete and variational circuits with different algorithms, such as tree-search with BFS, qsearch, etc. and allows for the implementation of custom gate sets. For comparison, we implement the trapped-ion gate set used in this work – see also Eqs. (1)-(3) – and run the compilation using the QSearch algorithm implemented in the aforementioned library, which is based on multiple tree traversal search algorithm such as the A^* algorithm [94] or the Dijkstra algorithm [95]. The results of these runs for the Toffoli gate are given in Table 1: we see that the A^* and Dijkstra routines give similar solutions, whereas the greedy heuristic proves significantly worse overall. The compilation routine offered by BQSKit, while significantly faster than the RL method, is unable to converge to the same compact solution discovered by the PS-LSTM algorithm.