

Reinforcement learning for ion shuttling on trapped-ion quantum computers

Maximilian Schier^{1,*}, Lea Richtmann^{2,*}, Christian Staufenbiel³, Tobias Schmale^{3,4},
Daniel Borcharding³, Michèle Heurs^{2,5,6} and Bodo Rosenhahn¹

¹*Institute for Information Processing (TNT), L3S, Leibniz University Hannover, Hannover, Germany*

²*Institute for Gravitational Physics, Leibniz University Hannover, Hannover, Germany*

³*QUDORA Technologies GmbH, Braunschweig, Germany*

⁴*Institute for Theoretical Physics, Leibniz University Hannover, Hannover, Germany*

⁵*Deutsches Zentrum für Astrophysik (DZA), Görlitz, Germany*

⁶*Deutsches Elektronen-Synchrotron (DESY), Zeuthen, Germany*



(Received 21 May 2026; accepted 6 August 2026; published 24 September 2026)

Scalable trapped-ion quantum computing is commonly realized with modular chips that feature distinct zones with specific functionalities, such as storage, state preparation, and gate execution. To execute a quantum circuit, the ions must be transported between these zones. This process is called ion shuttling. To achieve reliable computation results, the shuttling process must be optimized. However, as the number of ions increases, this becomes a high-dimensional optimization problem where optimal solutions cannot be computed efficiently. We demonstrate, to the best of our knowledge, the first use of reinforcement learning (RL) for the optimization of ion shuttling. RL is well suited for such scenarios, as it enables learning a strategy through direct interaction with the problem. We show that our RL approach outperforms current state-of-the-art heuristic techniques, yielding a reduction in shuttling operations of up to 36.3%. Furthermore, we show that our method is easily applicable to various chip architectures. Our approach offers a versatile method to study shuttling efficiency during chip design and, therefore, a highly relevant tool for future, more complex architectures.

DOI: [10.1103/b7ck-8wh4](https://doi.org/10.1103/b7ck-8wh4)

I. INTRODUCTION

Trapped ions are a leading platform for fault-tolerant quantum computing, combining long coherence times with high-fidelity gates [1]. Qubits are encoded in long-lived internal states and manipulated with laser or microwave fields [2–4].

To achieve all-to-all connectivity and enable scaling to large ion numbers, quantum charge-coupled device (QCCD) architectures were proposed [5,6]. These architectures provide distinct zones, specialized for specific operations, such as state preparation, storage, gate execution, and measurement. In addition to these specialized zones, QCCDs provide a mechanism for transporting ions between zones, a process called ion shuttling.

Despite relatively long coherence times, overall performance depends on circuit execution speed. Because shuttling is often the dominant latency, minimizing ion movements is critical [7–9]. The complexity of finding an optimal ion-shuttling schedule increases exponentially with the number of ions and circuit length. Furthermore, for every distinct

circuit, an optimal solution must be computed individually. Consequently, exact optimization is impractical at scale, and current compilers rely on heuristics [7,8,10–15].

While heuristics scale, they are often suboptimal and not generally transferable to hardware changes. This motivates learning-based approaches that discover improved schedules and adapt across devices. High-dimensional, sequential decision problems without labeled targets are well suited to reinforcement learning (RL), making RL a natural fit for shuttling optimization.

Our study targets architectures developed within Quantum Valley Lower Saxony (QVLS) [16]. We train an RL agent on arbitrary interaction circuits to minimize shuttling length, yielding a general-purpose scheduler applicable to any user input. Evaluated on algorithms from the Munich quantum toolkit (MQT) bench repository [17] and on quantum volume (QV) circuits [18], the agent achieves up to 36.3% fewer movements than a prior heuristic compiler [10] for problems with up to 50 ions, which is the current QVLS target capacity [19].

For small instances, we also report optimal solutions from a Boolean satisfiability (SAT) solver [20]. Our RL solutions are near optimal in most cases, and the remaining gap shrinks when allowing longer inference time. While the SAT solver requires compilation times that are not feasible for real applications, our RL agent finds solutions quickly.

To demonstrate adaptability to chip geometry, we also train our RL agent on different architectures, enabling direct comparisons of design choices and their impact on shuttling efficiency, thereby mimicking a shuttling-aware codesign process.

*These authors contributed equally to this work.

[†]Contact author: schier@tnt.uni-hannover.de

[‡]Contact author: lea.richtmann@aei.uni-hannover.de

These results show that RL can substantially improve ion shuttling for trapped-ion systems. It outperforms state-of-the-art heuristics, attains near-optimal performance where exact evaluation is feasible, generalizes across architectures, and supports shuttling-aware hardware design, reducing the reliance on hand-crafted heuristics as systems scale.

II. OVERVIEW OF THE ION-SHUTTLING PROBLEM

A. The quantum charged-coupled device

In trapped-ion quantum computing, qubits are encoded via the ion's different energy levels. To manipulate the ions, they are placed in traps where electromagnetic fields confine them. One approach is to line the ions up in a linear chain. However, linear chains are not scalable because controlling and cooling the vibrational modes in long chains becomes difficult [1,12]. These shortcomings led to the design of modular systems connecting multiple traps. Instead of being linear, two-dimensional devices are built in which ions are transported between different trap zones. Often, different zones are designated for specific operations. Typical zones include storage, compute (where quantum gates are executed), and readout. This approach is known as the QCCD architecture [5]. An important difference is that the ions now have to be transported between the zones depending on the actions to be performed on them. The actual ion shuttling on the chip is performed by varying the voltages applied to the DC electrodes, which moves the location of the axial trapping potential minimum, thereby transporting the trapped ion in the desired direction [21,22].

Executing a quantum circuit on a QCCD involves several steps. First, the circuit is translated to native gates and optimized to reduce the gate count, a process known as transpilation [10]. This yields a directed acyclic graph (DAG) of native operations consisting only of single- and two-qubit gates. The DAG encodes execution dependencies. To execute a native gate, the ions involved must be brought to the compute zone, so we require a protocol that shuttles ions through the chip's zones while preserving the circuit's logical order. Gates can be reordered according to their commutation relations. We only consider two-qubit gates as commuting if they act on distinct qubits, as in the reference methods. While this assumption is unnecessarily conservative in special cases, it allows to simplify the encoding and simulation. Single-qubit gates can be executed immediately before the two-qubit operation, once the ions are already in the compute zone. Accordingly, we temporarily remove all single-qubit gates before optimizing shuttling.

After a shuttling protocol is synthesized, the single-qubit gates are reinserted into the schedule. Throughout this work, we use “qubit” and “ion” interchangeably.

1. Example architecture 1: X-chip

Our first example architecture is the QVLS QROSS chip [23], the first proposal for a QCCD developed by QVLS [16]. It consists of four registers connected by an X-junction, we refer to it as the “X-chip.” The registers include a compute zone that can hold up to two ions, a state preparation and measurement (SPAM) zone that can hold only one ion, and two storage regions. An abstract sketch of this design is shown

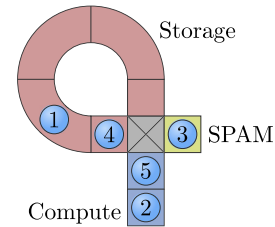


FIG. 1. Layout of the CIRQLE ion trap chip, also called Q-chip. The shown sketch has a single storage ring with a capacity of five qubits, which is connected to the other two registers with a junction. This ring acts like a carousel with empty or filled spots: It can only transport all stored qubits simultaneously through rotations. Some operations may only move empty spots through the junction. For example, in the given configuration, a clockwise rotation of the storage element does not move any qubit through the junction. It is likely that these operations are faster on real hardware than shuttling qubits through the junction. Our proposed method can handle this due to its semi-Markovian formulation, unlike other methods such as the SAT solver.

in the upper left corner of Fig. 2. The registers exhibit a stack-like behavior: When the ion closest to the junction is extracted from a zone and moved through the junction to another zone, the ion behind it is simultaneously moved next to the junction. This means that each time an ion crosses the junction, it counts as one shuttling step.

The maximum number of ions a storage register can hold determines the maximum number of qubits available for computation. Simulations for the X-chip were run with up to 50 ions; the first fabricated chip will support up to 24 ions [23].

2. Example architecture 2: Q-chip

We also study an alternative chip design, the QVLS CIRQLE chip [24], with a more compact storage register, allowing more ions to fit on the same chip size. Here, the storage zone is consolidated in a ring. The compute zone (capacity two ions) and the SPAM zone (capacity one ion) are connected to this ring via a junction, resulting in a Q-like shape, which is why we refer to it as the “Q-chip” (see Fig. 1). While in the X-chip, ions can be moved in and out of one storage zone independently of the other storage zone, in the Q-chip, we have only one common storage zone, and each movement will affect the location of all the other ions in the ring. The Q-chip storage ring works like a carousel with spots, some occupied, some empty. It can only rotate as a whole in either direction. To move an ion to a different zone, the ring is rotated until that ion is next to the junction and then transported to the destination zone.

Rotating about a single spot, whether it is filled or not, counts as one shuttling action. However, rotating empty spots through the junction is considered less costly than rotating ions through the junction, as there is no risk of ion loss. This nonuniform step duration is accounted for in our algorithm later.

B. Importance of optimizing the shuttling operations

The execution time of a quantum circuit is dominated by ion shuttling, which often accounts for more than half of the total run-time [7–9].

The main reason for reducing shuttling times is decoherence. While trapped ions offer relatively long coherence times—in the order of minutes for hyperfine qubits [1]—current transport operations still take milliseconds or microseconds [6,9,12]. The execution of large quantum circuits with high qubit counts can quickly exceed 10 000 shuttling operations on current QCCD architectures and can therefore take longer than the qubits' coherence times. Hence, to execute such circuits with high fidelity, optimized shuttling protocols are crucial.

To lower transportation overhead, several methods have been studied to physically reduce time requirements of individual transport operations [7,25–28]. However, increasing transport speed can, in the worst case, lead to ion loss. Besides this risk, the faster the transport, the greater the ions' energy, which in turn leads to heating. This makes it more difficult to address the ions for gate execution and is detrimental to the gate fidelity [8,11–13,28]. To remove the energy acquired, an extra cooling step can be implemented [9]. Still, for achieving high gate fidelities, the transport speeds are limited, and the amount of transport steps should therefore be reduced as much as possible [5,11].

Additional decoherence can occur during transport due to inhomogeneous magnetic fields [1]. While modern shuttling techniques mitigate these effects, they remain a potential source of error; optimizing the shuttling schedule reduces the qubits' exposure time [7,9].

Overall, reducing the shuttling times is widely considered crucial for achieving reliable trapped-ion quantum computing with QCCDs [1,9].

C. State-of-the-art shuttling techniques

As the amount of available qubits in current QCCDs increases, it is usually not possible to develop scalable methods that are able to find an optimal solution to minimize the shuttling operations [11]. Methods for finding shuttling protocols, therefore, often rely on sorting techniques [6,7] and on heuristics to determine which ions are shuttled in what order. Current strategies for ion shuttling differ widely in terms of the types of architectures they study (amount of storage and gate zones, connections between these zones, possibility to perform parallel operations) and shuttle operations they include (linear shifts, movements through junctions, splitting and combining ion crystals, and swapping the positions of two ions). As a summary, we give here an overview of the main types of heuristics that are applied:

Determination of possible tasks. This refers to the heuristic where, instead of looking at all the tasks, i.e., gates to be executed, the compiler focuses on the tasks that can actually be performed, making the action space smaller [12]. Approaches often use a dependency graph in order to determine the order in which gates have to be executed [7,11,13] and to select which gate should be executed next [10,14].

Least movements. This group of heuristics prioritizes gates that can be executed quickly, when one ion is already in the compute zone or moves ions first that are close to the compute zone [7,10,13].

Storage selection. This heuristic refers to making selective use of different storage regions: Reference [13] moves qubits that are not needed in the near future to a designated storage location so they do not block the path. In a similar idea, Ref. [7] stores ions that are used together in proximity. Reference [10] also implements proximity sorting and uses the readout zone as temporary storage to make ions needed shortly after easily accessible.

Handling traffic blocks. Different heuristics are used in order to avoid an ion from being blocked because the path it has to take is occupied by other ions [13]. This is done, for example, through specific movement combinations like splitting and merging [11,12] or by moving ion chains on closed paths together in loops [14,29,30] for architectures where this is possible.

Reference [15] proposes a probabilistic formula that accounts for several heuristics, such as *least movements* and *handling traffic blocks*. With this formula, they can then determine whether an ion should move or stay in the same trap.

The mentioned strategies are planning strategies; a shuttling protocol is developed for a given circuit in advance and then executed on the actual QCCD. In contrast, Ref. [9] uses real-time compilation. They also employ heuristics, such as allocating storage regions and accounting for movement costs.

For the sake of concreteness, we compare our proposed RL-based approach against two implementations of the above-mentioned methods: the heuristics-based compiler based on Ref. [10] and the SAT-based approach from Refs. [20,31]. We chose these two approaches for our comparisons because they have been used so far for the QVLS X-chip.

1. Reference 1: heuristic compiler

In the framework of the QVLS X-chip, a shuttling compiler was developed to address the ion-shuttling problem using heuristics derived from observations of the chip's architecture [10]. The challenge of orchestrating ions across the chip to execute a given quantum circuit was therefore decomposed into several phases, two of which are particularly relevant to this paper.

Graph serialization. The compiler begins by representing the input quantum circuit as a DAG, which encodes a partial order for gate execution. This partial order allows for flexibility in scheduling, enabling the compiler to select an execution sequence that minimizes the number of ion-shuttling operations. A custom algorithm generates n_p distinct sequences of up to n_g interaction gates (i.e., two-qubit gates), each respecting the partial order. The sequence requiring the fewest shuttling operations (determined via the subsequent compiler stages described below) is selected for execution, removed from the DAG, and executed. This process iterates until the DAG is empty.

Ion orchestration. Given a selected sequence of n_g interaction gates, the ion orchestration module generates a sequence of shuttling directives to move ions across the chip. The primary objective is to repeatedly position ions into the compute zone such that the n_g gates can be executed in the prescribed order. Different heuristics are employed to guide the

generation of efficient shuttling sequences. For details on the heuristics, see Ref. [10].

Increasing the size of the intermediate gate sets (n_g) allows the compiler to consider more future gates, thereby reducing the total number of shuttling operations. However, this comes at the cost of increased compilation time. For this work, we set $n_g = 4$, which provides a favorable trade-off between short shuttling sequences and reasonable compile time.

This work employs a version of the heuristic-based compiler developed further by QUDORA Technologies GmbH. While minor enhancements have been made to the original release, the core structure and heuristics of the shuttling compilation process remain unchanged compared to the original publication [10].

2. Reference 2: SAT solver

For benchmarking purposes, it is useful to compare obtained trajectories against optimal ones. While finding these optimal trajectories is likely infeasible in the general case, we can at least study small instances to gain some basic insight.

In principle, a naive exhaustive search through all shuttling sequences of a fixed length could be used to find the shortest option for a given input circuit; however, a simple estimate shows that this becomes infeasible quite quickly: For some of the QCCD architectures studied below, there are at least three possible next shuttling operations from any given ion configuration. For some of the below circuits, we find optimal shuttling sequences consisting of more than 70 steps, yielding a total of more than $3^{70} \approx 10^{33}$ shuttling sequences that the naive search would have to consider. Reference [20] suggests to ameliorate this situation by formulating the ion-shuttling problem as a boolean satisfiability problem (SAT) and leveraging the decades-long efforts that have gone into the development of automated solvers for such problems (“SAT solvers”).

The core idea is to define boolean variables $b_{i,x,t}$ that indicate whether an ion $i \in \{0, \dots, N\}$ is located at location x at time step $t \in \{0, \dots, T\}$, for some upper limit on the number of time steps T and ion count N . For a given circuit and chip geometry, one can then construct a boolean function $\Phi_T : \{b_{i,x,t}\} \rightarrow \{\text{True}, \text{False}\}$ that encodes whether the input variables $\{b_{i,x,t}\}$ encode a valid shuttling sequence for executing the circuit. For details on how Φ_T is constructed, see Refs. [20,31]. Finding such a set of variables that makes Φ_T evaluate to “True” (“satisfying assignment”) is precisely an instance of a SAT problem. In such a representation, existing solvers can be used to check whether a satisfying assignment exists for a particular value of T . If the solver indicates that Φ_T is not satisfiable (i.e., no valid shuttling sequence of length T exists), T may be incremented until a solution is found. If this procedure is started from $T = 1$, a provably optimal solution will be found. From a satisfying assignment, a shuttling schedule can then be reconstructed.

This approach is not scalable, since SAT is complete on the complexity class of nondeterministic polynomial-time decision problems (NP-complete) in general [32]. However, we find this approach to work very well in practice for small problems, allowing us to optimally compile instances unapproachable using naive exhaustive search.

D. Motivation for using reinforcement learning for ion shuttling

As explained in Sec. II B, it is paramount for reliable trapped-ion quantum computing to reduce the ion-shuttling overhead. For large numbers of ions, this becomes a high-dimensional optimization problem, making it impractical to find optimal solutions. This is why, as shown in Sec. II C, heuristics are employed. However, by their nature, heuristic approaches cannot comprehensively cover every eventuality, and efforts to achieve such completeness lead to a disproportionate increase in development effort. High-dimensional optimization problems that are intractable for conventional optimization methods are a classical use case for machine learning. Because the optimal solution is unknown, labeled data are unavailable, and supervised learning is not applicable. Reinforcement learning, on the other hand, is well suited: An agent takes sequential actions, observes the resulting state, and receives a reward reflecting progress toward the objective. Initially, the agent explores randomly. It updates its policy, that is, its rules for decision making, to maximize its cumulative reward and progressively learns efficient strategies. It is therefore also capable of finding patterns that were not thought of theoretically beforehand. By training the RL agent on a diverse set of randomly generated circuits, it learns to generalize and compile arbitrary circuits without retraining.

Reinforcement learning has been used for a wide variety of scheduling or combinatorial optimization problems, such as job-shop scheduling [33], the vehicle routing problem [34], and has also been used to optimize compiler phases for classic computers [35]. For compilation in trapped-ion quantum computing, RL has been used previously to optimize the transpilation process [36]. However, to the best of our knowledge, this is the first work to employ it for the optimization of ion shuttling.

III. PROPOSED REINFORCEMENT LEARNING METHOD

In this section, we present the relevant concepts of reinforcement learning as well as the important technical details of our proposed RL ion-shuttling compiler, which we call “RLIonS.”

A. Problem formulation

We formulate ion shuttling as a semi-Markov decision process (SMDP) [37]. Like a regular Markov decision process (MDP) [38], the SMDP models the interaction of an agent with a system at certain time points, the discrete “events.” Unlike an MDP, an SMDP permits the system to behave non-Markovian while transitioning between time points under continuous time. More importantly, the SMDP formulation allows that steps between two events have nonuniform duration, whereas the standard MDP does not. This is an important aspect when modeling an arbitrary ion-shuttling chip, as some shuttling operations may take longer than others.

Formally, an SMDP is defined as the tuple $(\mathcal{S}, \mathcal{A}, A, P, F, R, \beta)$.

(1) The state space $\mathcal{S}$ is the discrete space of all possible configurations of the ion-shuttling simulation. This entails the location of all qubits on the trapped-ion chip and the pending two-qubit gates.

(2) The action space $\mathcal{A}$ is the discrete space of the ion movements the chip can perform in general.

(3) The action mask $A : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A}) \setminus \{\emptyset\}$ describes the permissible subset of actions for a given state. Here, $\mathcal{P}$ is the power set.

(4) The transition dynamics $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ describe the new state at the next decision epoch after performing an action from a given state. Transitions are deterministic for ion shuttling.

(5) The decision epoch duration $F : \mathcal{S} \times \mathcal{A} \rightarrow (0, \infty)$ describes the duration until the next decision epoch, i.e., the duration until the agent selects the next action. It is dependent on the current state and action and is deterministic for our problem.

(6) The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ describes the immediate reward for a transition, i.e., a feedback on how beneficial the selected action is during the current decision epoch.

(7) The discount rate $\beta \in (0, \infty)$ describes how future rewards are valued. It is linked to the discount factor of a standard MDP through $\gamma = e^{-\beta}$ assuming a decision epoch duration of 1.

A (probabilistic) policy is any function of the form $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, where Δ is a probability simplex. A trajectory ζ is a sequence of states and actions that can be obtained by rolling out the interaction of a policy and the system described by the SMDP: $\zeta = (s_0, a_0, s_1, a_1, \dots)$. The initial state s_0 is usually sampled from a distribution of valid start states $s_0 \sim \rho$, with $\rho : \Delta(\mathcal{S})$. Performing a rollout using a policy π is commonly notated in short form $\zeta \sim \pi$. The total discounted return G of a trajectory, i.e., the discounted sum of the individual step rewards, is defined as

$$G(\zeta) = \sum_t e^{-\beta \cdot \sum_{u=0}^{t-1} F(s_u, a_u)} R(s_t, a_t, s_{t+1}). \quad (1)$$

Given such an SMDP, the goal is finding the optimal policy π^* over the expected discounted return J :

$$\pi^* = \arg \max_{\pi} J(\pi), \quad (2)$$

$$\text{where } J(\pi) = \mathbb{E}_{\zeta \sim \pi} [G(\zeta)]. \quad (3)$$

Since the goal in ion shuttling is finding the shortest valid shuttling sequence, the reward term should continuously penalize at a constant rate c_r for the entirety of a decision epoch:

$$R(s, a, s') = \int_0^{F(s,a)} -c_r \cdot e^{-\beta t} dt = c_r \cdot \left[\frac{e^{-\beta t}}{\beta} \right]_0^{F(s,a)} \quad (4)$$

$$= c_r \cdot \frac{e^{-\beta F(s,a)} - 1}{\beta}. \quad (5)$$

For simplicity of notation, we have defined the transition dynamics P and the decision epoch duration F to be deterministic. However, this is not required for our proposed method, and future work could employ nondeterministic functions, for example, to model nonideal effects such as noise.

B. Proximal policy optimization

A parametrized policy is defined as $\pi_{\theta} : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, with parameters θ . The policy gradient theorem [39] specifies that

π_{θ} converges locally by gradient ascent on

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{s \sim d^{\pi_{\theta}}, a \sim \pi_{\theta}(\cdot|s)} [\Psi^{\pi_{\theta}}(s, a) \nabla_{\theta} \ln \pi_{\theta}(a|s)], \quad (6)$$

if $\Psi^{\pi_{\theta}}$ is a bias-free estimator of the state-action values or advantages of π_{θ} . Here, $d^{\pi_{\theta}}$ is the discounted state distribution under policy π_{θ} . This objective requires that all sampled trajectories originate from the current policy π_{θ} . Thus, after every gradient step, an often impractically large number of trajectories must be collected for the next step.

Proximal policy optimization (PPO) [40] is a commonly used policy gradient method that reduces the number of samples per gradient step. First, it is allowed that samples used for training are collected by a close policy, but not exactly the same policy. This is corrected using importance sampling and clipping. Let θ' be an old close parametrization of π_{θ} . The objective of PPO is defined as

$$\nabla_{\theta} J(\pi_{\theta}) = \nabla_{\theta} \mathbb{E}_{s \sim d^{\pi_{\theta'}}, a \sim \pi_{\theta'}(\cdot|s)} \left[\min \left(\frac{\Psi^{\pi_{\theta'}}(s, a) \pi_{\theta}(a|s)}{\pi_{\theta'}(a|s)}, \Psi^{\pi_{\theta'}} \text{clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta'}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) \right) \right], \quad (7)$$

where ϵ is the clipping ratio. Second, the estimator $\Psi^{\pi_{\theta'}}$ of PPO is biased. PPO uses a combination of the bias-free estimator given by the Monte Carlo rewards and the biased estimator of a parametrized model $V_{\phi} : \mathcal{S} \rightarrow \mathbb{R}$. For a regular MDP, PPO uses the generalized advantage estimator (GAE) $\hat{A}_t$ [41] as $\Psi^{\pi_{\theta'}}$, which is defined as

$$\hat{A}_t = \delta_t + (\gamma \lambda) \hat{A}_{t+1}, \quad (8)$$

$$\text{where } \delta_t = R(s_t, a_t, s_{t+1}) + \gamma V_{\phi}(s_{t+1}) - V_{\phi}(s_t). \quad (9)$$

For an SMDP, the GAE is modified to

$$\hat{A}_t = \delta_t + (\lambda e^{-\beta})^{F(s_t, a_t)} \hat{A}_{t+1}, \quad (10)$$

$$\text{where } \delta_t = R(s_t, a_t, s_{t+1}) + e^{-\beta F(s_t, a_t)} V_{\phi}(s_{t+1}) - V_{\phi}(s_t). \quad (11)$$

C. Representation

A trapped-ion chip can have a geometrically complex shape, but ultimately the ions acting as qubits are located at positions from a discrete set. We can therefore express the state of a given chip in tabular form. Let n_{cells} be the number of positions any ion can be located at, which we call a “cell.” The configurations of any chip are then expressed by a mapping or table from cells to qubits $K : \{1, \dots, n_{\text{cells}}\} \rightarrow \{\diamond, 1, \dots, n_{\text{max}}\}$, where $\diamond$ indicates an empty cell and n_{max} is the maximum number of ions supported by a given chip. A valid mapping should also have no duplicate qubits: $\forall i, j : i \neq j \Rightarrow K(i) = \diamond \vee K(i) \neq K(j)$. How many cells exist and how they are mapped to the physical locations of the chip depend on the chip at hand, but it is usually straightforward. An example for the QVLS X-chip is shown in Fig. 2. Next, we assume that a circuit of two-qubit gates is given as a sequence of gates $p = (g_1, g_2, \dots)$, with gate $g_i = (x_i, y_i)$. Thus, x_1 is the first operand of the first gate, and y_1 is the second operand of the first gate, and so on. Gates do not have to be executed in sequence, as long as the gate to be executed has no operand used in a prior gate that still needs to be executed.

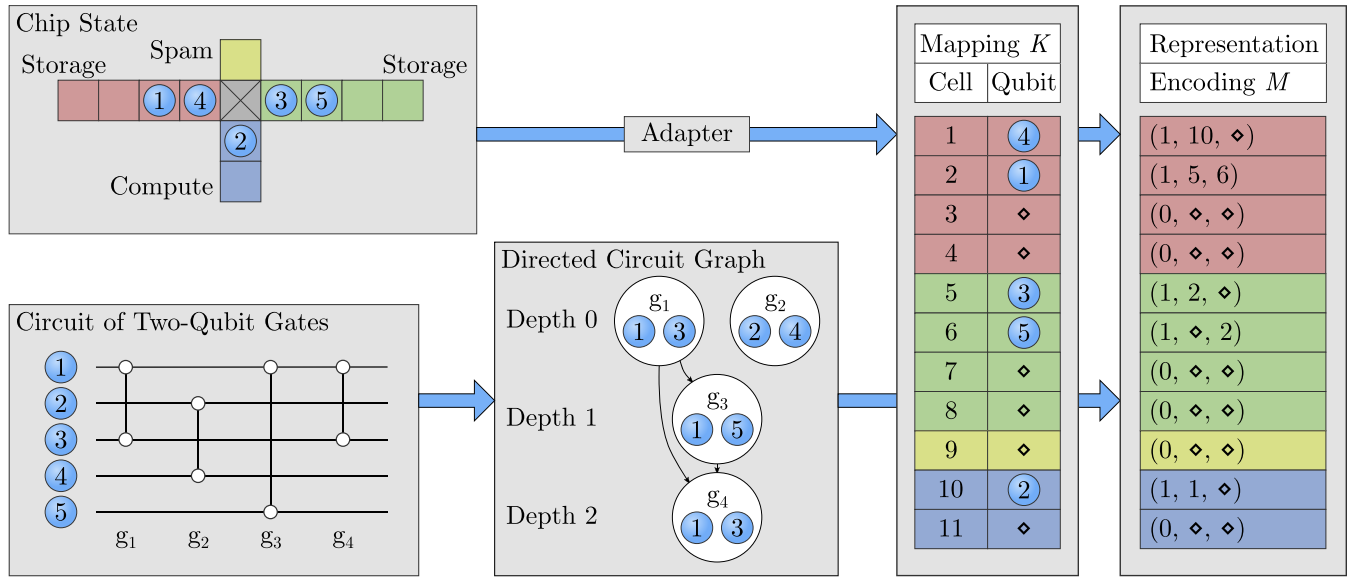


FIG. 2. Proposed representation. The top left shows the chip state. A chip-specific adapter translates the chip state into a cell-qubit mapping, shown in the mapping table K on the right side. The rest of the observation encoding is chip agnostic. The directed acyclic graph of two-qubit gates (middle) is constructed from the circuit (bottom left). The depth of each gate is determined. Finally, the encoding table M is constructed. Each row refers to a cell on the given chip. The first entry per row indicates the presence of a qubit (0 if the cell is empty). The next entries within a row indicate the cell location of the other operands at increasing depth up to the lookahead. The special value $\diamond$ indicates no gate at this depth. For example, consider the encoding of cell 6. This cell holds qubit (5), which is used only in gate g_3 located at a depth of 1. That cell's encoding is (1, $\diamond$, 2), because it has a qubit present (1 in first entry); that qubit has no gate at depth 0 (therefore the $\diamond$ in second entry), but it does have a gate at depth 1, which uses the qubit (1) as the other operand, which is located in cell 2 shown in the third entry. Depth 2 is not considered for this example with $k_{\text{lookahead}} = 2$. Note that in our training we implement a lookahead of 4.

1. Requirements for representations

When employing a neural network as a policy π , the state space $\mathcal{S}$ must usually be transformed to compatible representations, e.g., vectors, using some transformation. For simplicity, in a slight abuse of notation, we use $\mathcal{S}$ as the representation space directly. A good representation of the chip state and circuit to be executed should be invariant to changes that have no influence on the validity of any given trajectory and the return under that trajectory and are thus equivalent regarding the solution. In the RL literature, such a relation of states is called a bisimulation relation [42]. Let B_M be the maximal bisimulation relation such that the partition $\mathcal{S}/B_M$ has lowest cardinality. A good state representation $\mathcal{S}$ should minimize $|\mathcal{S}| - |\mathcal{S}/B_M|$. Otherwise, an agent must first learn that an infeasible large number of representations all represent the same underlying state.

Invariance to relabeling. First, a good representation should be invariant to a relabeling of the qubits. Such a relabeling can be carried out by swapping the labels of two qubits in K and also swapping all occurrences in p .

Example. Consider an arbitrary chip state with 50 qubits. There are $50! \approx 3 \times 10^{64}$ unique relabeling permutations of the state.

Conclusion. A good representation should not directly contain the label of a qubit.

Invariance to commutative circuit reordering. As described above, in the sequence of two-qubit gates p , the first gate does not need to be executed first, as long as another gate after it has two operands that do not appear before that gate. Thus, any two neighboring gates $g_i = (x_i, y_i)$ and

$g_{i+1} = (x_{i+1}, y_{i+1})$ can be swapped and the resulting new circuit is equivalent, as long as $x_i \neq x_{i+1} \wedge x_i \neq y_{i+1} \wedge y_i \neq x_{i+1} \wedge y_i \neq y_{i+1}$.

Example. Consider the QV(50) problem. There are 50 independently generated layers of 25 two-qubit gates each. Within a layer, all two-qubit gates are commutative. Due to the problem structure, there are at least $(25!)^{50} \approx 3 \times 10^{1259}$ equivalent circuit permutations for any given start state.

Conclusion. A representation should not directly contain the sequence index i of a two-qubit gate g_i .

2. Proposed representation

The core idea of our proposed representation is abstracting the qubit label and sequence position of a two-qubit gate by encoding a gate through the cell location of the other operand and the depth of the gate in the dependency graph. This is illustrated in Fig. 2 for a lookahead of $k_{\text{lookahead}} = 2$. The following steps are performed:

(1) A chip-specific adapter translates the chip state (top left) into a tabular form K (columns “cell” and “qubit” on the right). In our case, the adapter simply iterates all zones starting with the position next to the junction.

(2) If the circuit (bottom left) is given as a list of gates, the directed acyclic graph of the circuit is calculated first (bottom center). The depth of a gate is defined as the length of the longest path on the transpose graph starting from the node of that gate.

(3) The encoding matrix M is computed (right). For each cell, it is encoded whether it is occupied by a qubit. Next, for depths in $\{0, \dots, k_{\text{lookahead}} - 1\}$, it is checked if a gate at

that depth exists with the qubit of the current cell. If it exists, the cell of the other operand is encoded. Otherwise, an empty token $\diamond$ is encoded.

Gates at a depth of $k_{\text{lookahead}}$ or larger are not included in the representation. This is the case for g_4 in Fig. 2. The final observation input for the agent is constructed by flattening M and appending the total remaining gate count. We then employ a sinusoidal embedding [43] to encode numeric values (see Appendix C for details).

By abstracting qubit labels and encoding gate dependencies via cell location, we avoid the growth of the state space by a factorial of the ion number (see Examples in Sec. III C 1) and thereby drastically reduce the number of possible representations.

D. Network architecture

Both the policy and value functions are implemented as dense neural networks with residual feedforward blocks. Such a residual feedforward block of width h applies in sequence a ReLU activation, a fully connected layer with h units, another ReLU, and another fully connected layer with h units. Finally, in the residual feedforward block, the initial input signal is added to the output again. Policy and value networks share no parameters. Per network, we use one fully connected layer to project onto the hidden feature size n_{hidden} . Next, n_{blocks} residual blocks of width $h = n_{\text{hidden}}$ are applied in sequence. Finally, another ReLU activation and a last fully connected layer is applied. This last layer has one output for the value function and n_{act} for the policy. This design is selected as residual connections are a common choice in current RL literature [44,45].

E. Description of training process

1. Shaped reward

The basic reward signal for a goal-reaching problem is very sparse, as the agent receives a negative reward at a constant rate c_r until a goal state is reached. If the problem only terminates upon reaching a goal state and the agent has not encountered any goal states yet, the value of every state must be estimated as $V = -c_r \int_0^\infty e^{-\beta t} dt = -\frac{c_r}{\beta}$. The resulting optimal policy until a goal state is encountered is a uniform distribution over the allowed actions in every state. On complex problems, this is a very poor exploration policy due to the high branching factor of the graph and the resulting large number of states searched while lacking any guidance. To give the agent some prior knowledge, a potential-based shaped reward can be introduced. For any standard MDP, a surrogate MDP can be constructed with any reward function of the form

$$R'(s, a, s') = R(s, a, s') + \gamma_s \phi(s') - \phi(s), \quad (12)$$

where R is the original reward, $\gamma_s = \gamma$ is the original discount factor, and ϕ is some potential over the state space that is 0 for all terminating states: $\forall s \in \mathcal{S}_T : \phi(s) = 0$. When these conditions are met, any policy which is near optimal on R' is also near optimal on the original R [46]. For the SMDP, we modify the function to account for the step duration:

$$R'(s, a, s') = R(s, a, s') + \gamma_s^{F(s,a)} \phi(s') - \phi(s). \quad (13)$$

Let $f_{\text{gates}}(s)$ be the number of gates that still need to be applied in state s . Then, our proposed potential-based shaped

reward is

$$\phi(s) = -f_{\text{gates}}(s). \quad (14)$$

If γ_s is selected differently from γ , then a policy that is near optimal on the surrogate problem is not necessarily near optimal on the original problem. We set $\gamma_s > \gamma$, making the policy greedy in improving on the potential ϕ . Greedy in this context means that the agent prefers more immediate reward. This is actually required if the effective time horizon induced by γ is too short to solve the problem, which can be the case for large quantum circuits as we show in our experiments (see Sec. IV E).

2. Problem generation during training

When training the RL agent, a diverse range of starting states is desirable, such that the entire possible problem space is well covered. A starting state is generated by first drawing the number of ions or qubits on the chip: $z \sim \text{Uniform}(\{2, \dots, n_{\text{max}}\})$. Here, n_{max} is the maximum number of ions supported. The qubits are positioned on a random storage element. Next, we sample on average $\frac{z}{n_{\text{max}}} \cdot n_{\text{gates}}$ gates, where n_{gates} is selected to reflect the number of gates expected in a large circuit. Per gate, the first operand $x_i \sim \text{Uniform}(\{1, \dots, z\})$ is sampled from the available qubits, then the second without replacement: $y_i \sim \text{Uniform}(\{1, \dots, z\} \setminus \{x_i\})$.

F. Description of inference

Compilation of new circuits using a trained model is straightforward. Given the trained policy π , the starting state s_0 is constructed from the initial chip state and the initial circuit to be compiled. Let $F_{\text{total}}(\zeta)$ be the total duration under the trajectory ζ . The compilation now consists of sampling a trajectory $\zeta \sim \pi$ from the starting state s_0 under policy π . Because this process is probabilistic, it can be repeated as often as desired and also be parallelized to achieve better results. If repeated, the solution is that trajectory ζ with the smallest $F_{\text{total}}(\zeta)$. This approach has two advantages. First, it allows compilation in real time, since the next action can be sampled from the policy in constant time for any state. More importantly, under an increasing time budget the compiler can generate increasingly efficient shuttling sequences (see Sec. IV C).

IV. RESULTS

We train our proposed method RLIonS as described in Sec. III using the hyperparameters from Appendix A for up to 50 ions for each chip architecture: the X-chip and two variants of the Q-chip as described in Sec. IV D. To ensure comparability, we use the same initial qubit-to-ion mapping as the heuristic compiler. We do not tune hyperparameters per experiment such that the general applicability of our method can be evaluated.

A. Main experiment on MQT bench circuits

To compare the results from our reinforcement learning method RLIonS with those of the exact SAT solver and the heuristic approach, we use the MQT bench dataset [17]. This dataset contains well-known quantum circuits that are often

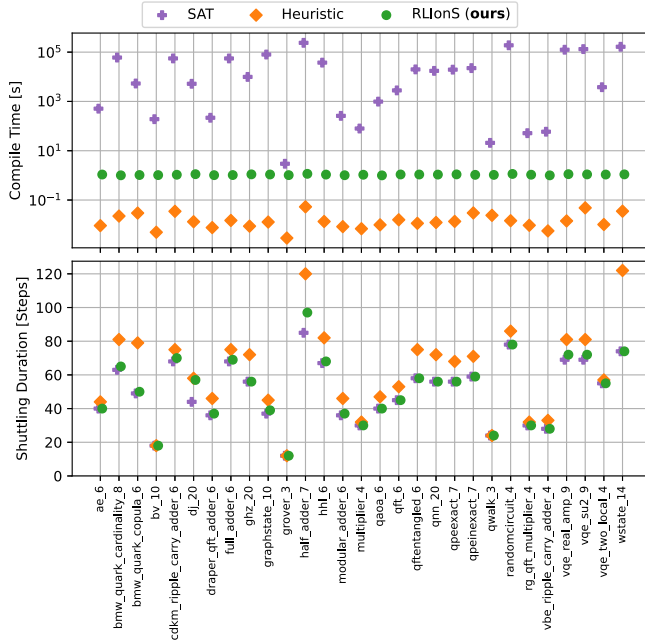


FIG. 3. Comparison of ion-shuttling durations using trajectories optimized by different methods on the QVLS X-chip. For each type of problem within the MQT dataset, the largest example (the number at the end of the problem name indicates the qubit number) that could still be optimized by the SAT solver is shown. On average, the shuttling sequences generated by our proposed RL compiler are 2.6% worse than the optimal solutions of the SAT solver and 14.0% better than those of the heuristic compiler. The SAT solver requires 12 h per problem on average, our approach 1.1 s.

used in other algorithms and serve as established benchmarks. It includes quantum circuits such as Grover’s algorithm, the quantum Fourier transform (QFT), quantum approximation optimization algorithm (QAOA), and others. For each circuit, the number of qubits can be chosen. For our analysis, we use target-independent MQT circuits with optimization level 3 and include all circuits that we can transpile to our native gates, and for which at least the smallest instance is solvable with the SAT solver. As mentioned before, finding an exact solution breaks down when the problem dimension gets too big. Here, we terminated solver runs if the wall time exceeds 10 h for one boolean function Φ_T . The maximum qubit number the SAT solver could handle varied between 3 and 20 qubits within the MQT bench problems. In the majority of the cases, it stayed far below 20 qubits. We compare our proposed RLionS compiler on the X-chip with the SAT solver and heuristic compiler. While training for up to 30 ions would be sufficient for this experiment, we have used the same trained agent on the X-chip for all experiments to show that, once trained, RLionS can solve a variety of problems.

For each circuit type, we select the instance with the largest number of ions for which the SAT solver was still able to find a solution when shuttling on the X-chip. Figure 3 shows, for this set of circuits, the compile time (upper chart) and the resulting shuttling duration in steps (lower chart) for the three methods. RLionS outperforms the heuristic compiler for all circuits and has, for most circuits, a similar or only slightly worse performance than the exact SAT solver. On average,

TABLE I. Comparison of ion shuttling durations using trajectories optimized by the heuristic compiler and our proposed method on the X-chip for the largest version per type of problem within our dataset of the MQT circuits. On average, our proposed compiler generates 19.8% faster shuttling sequences.

Problem	RLionS steps	Heuristic steps
ae_15	319	356
bmw_quark_cardinality_20	186	242
bmw_quark_copula_20	678	845
bv_10	18	18
cdkm_ripple_carry_adder_20	309	332
dj_20	57	58
draper_qft_adder_20	493	668
full_adder_20	309	332
ghz_32	92	120
graphstate_20	107	149
grover_7	3081	3499
half_adder_19	744	1118
hhl_20	1179	1519
hrs_cumulative_multiplier_17	4122	4625
modular_adder_20	493	668
multiplier_20	2879	4675
qaoa_20	835	1167
qft_20	609	715
qftentangled_30	1448	1855
qnn_20	56	72
qpeexact_20	577	700
qpeinexact_20	585	700
qwalk_7	3377	3957
randomcircuit_20	4355	5681
rg_qft_multiplier_20	2914	4675
vbe_ripple_carry_adder_19	361	448
vqe_real_amp_20	186	246
vqe_su2_20	186	246
vqe_two_local_20	1898	2622
wstate_20	110	122

of the circuits that were solvable for the SAT solver, RLionS performs only 2.6% worse than optimum. To put this result into context, it is important to look at the compile time: While RLionS (1.1 s) and the heuristic compiler (<0.1 s) are very fast, the SAT solver takes on average 12 h to find its exact solution. This timescale is not feasible for actual application. Furthermore, as explained in Sec. IV C, the optimality gap can be further closed by increasing RLionS’ compile time slightly, while still being much faster than the SAT solver and remaining within an acceptable compile-time regime for the actual use case.

For further comparison, in Table I, we select the largest version of each circuit type in our dataset and compare RLionS to the heuristic compiler. Both methods find shuttling schedules for all of these problems. On average, RLionS takes 19.8% fewer steps than the heuristic compiler.

Additionally, the results for all studied MQT circuits, as well as animations of some example shuttling sequences are presented in the Supplemental Material [47].

A common practice in RL is to evaluate on the same domain and distribution that were used for training. A good

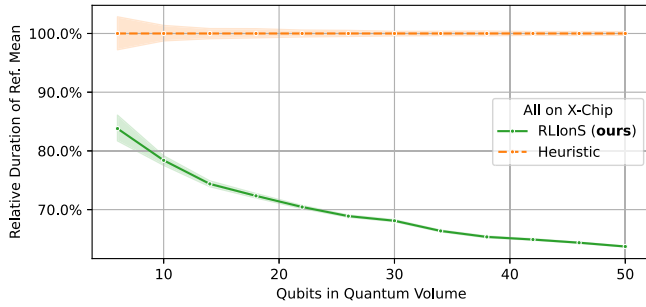


FIG. 4. Comparison of ion-shuttling duration of our proposed method to the reference heuristic compiler on the QVLS X-chip for increasingly difficult quantum volume circuits. Shaded areas indicate 95% confidence intervals. With a comparable inference time budget, our proposed compiler finds significantly more efficient ion-shuttling trajectories. The advantage of our approach increases from 16.2% on small circuits up to 36.3% on the largest ones.

shuttling compiler, however, should perform well on arbitrary circuits. Because our training circuits are generated by uniformly sampling random gates, complex MQT circuits with specific gate patterns were unlikely to appear during training. It is therefore a crucial finding that our agent generalizes effectively to previously unseen problems.

B. Main experiment on quantum volume circuits

Another widely used benchmark is the QV circuits, which were developed to compare the quality of quantum computers across different platforms [18]. They are random square-shaped circuits, meaning the number of qubits equals the circuit depth. In each layer, the qubits undergo a random permutation, after which two-qubit gates are applied to neighboring qubits. As these circuits have a well-defined structure and can be generated for any number of qubits, we use them to evaluate RLIonS' performance for increasing qubit counts. For this experiment, we compare our RLIonS compiler with the heuristic compiler on the X-chip. The SAT compiler is not included here as it does not scale beyond six qubits; see Sec. IVC for a detailed comparison on QV(6) circuits.

Figure 4 shows RLIonS' resulting shuttling duration relative to the duration obtained by the heuristic compiler on the

X-chip for qubit numbers between 6 and 50. For comparable results, the compilation with RLIonS is stopped if it exceeds the maximum of the compile time of the heuristic compiler and 1 s. RLIonS achieves much faster shuttling schedules, yielding an advantage of up to 36.3% for large problems that require more than 10,000 shuttling steps. It is especially important that the speed-up the reinforcement learning method provides increases with increasing ion number. This makes our method particularly useful, as the problem of executing long quantum circuits within the ions' coherence time becomes more pronounced as qubit numbers increase.

C. Scaling with time during inference

In Sec. III F, we have outlined how the probabilistic rollout during compilation can be used to generate increasingly good solutions with increasing time. We have analyzed this scaling on quantum volume problems with six qubits in Table II. Our proposed method is simulated with a time budget scaling from 100 ms up to 100 s. This is done by compiling 100 predetermined QV(6) circuits for 600 s per circuit with RLIonS on the X-chip. Then, bootstrapping is used to draw 20 000 samples of the respective budget, e.g., 100 ms, to determine the average metrics reported in the table. A comparison with the heuristic compiler and SAT solver on the same 100 circuits is included. The optimality gap is reported, which is the difference in normalized steps between the proposed solution and the optimal solution. For each tenfold increase in inference time, our method can close the average optimality gap further, going from a difference of 0.71 steps for 0.1 s compilation time to only 0.42 steps for 100 s compilation time. These findings highlight the versatility of our proposed approach, initially finding already very good solutions compared to the heuristic compiler, which can further be improved with moderate increases in compile time compared to the SAT solver.

D. Extension to other architectures

So far, our proposed RL ion-shuttling approach has been applied to a chip with an X-junction and two separate storage stacks. We now evaluate how well our method generalizes to other architectures. We focus on a Q-chip design as shown in Fig. 1 and discussed in Sec. IIA 2. It is difficult to know *a priori* how different design choices will affect shuttling effi-

TABLE II. Optimality gap (absolute difference in steps to the optimal solution) on 100 randomly generated quantum volume problems with six qubits. The optimal solutions were computed using a SAT solver and have an average of 48.52 steps. With a larger inference time budget, the average optimality gap of our method closes and our method is more likely to find the optimal solution. With a time budget of 10 s, RLIonS always proposes solutions within two steps of the optimum. As the SAT solver is the ground truth, it is always optimal. However, it requires more than 3 h per problem on average for these relatively small circuits. The heuristic compiler is as fast as when running our method with the lowest time budget, but has a larger optimality gap than our proposed method and is never within two steps of the optimal solution.

Method	Compile time per problem	Optimality gap				Average opt. gap
		0 steps	1 step	2 steps	> 2 steps	
RLIonS (ours)	0.1 s	47.24%	36.28%	14.66%	1.82%	0.71 steps
RLIonS (ours)	1.0 s	51.28%	36.36%	12.32%	0.04%	0.61 steps
RLIonS (ours)	10.0 s	57.52%	34.08%	8.40%	0.00%	0.51 steps
RLIonS (ours)	100.0 s	62.32%	33.08%	4.60%	0.00%	0.42 steps
Heuristic	0.1 s	0.00%	0.00%	0.00%	100.00%	10.65 steps
SAT	12766.5 s	100.00%	0.00%	0.00%	0.00%	0.00 steps

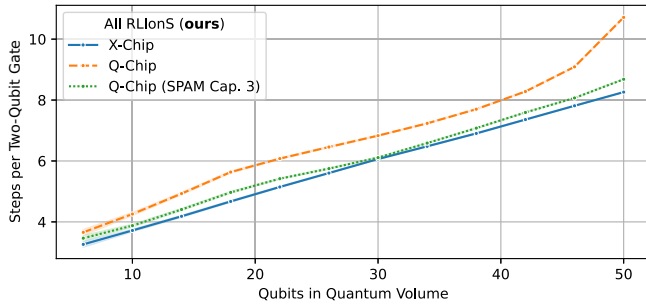


FIG. 5. Ion-shuttling duration for different architectures optimized with our proposed RL method. Shaded areas indicate 95% confidence intervals. All tested architectures are consistently optimized by our method. The generated shuttling sequences for the standard Q-chip use significantly more steps per two-qubit gate than the X-chip for all tested problem sizes. This can be mitigated by increasing the capacity of the SPAM zone from one to three qubits. Such exploratory design studies are very easy to perform using our method, as it does not require the development of new heuristics or other rules for new hardware.

ciency. It is therefore invaluable to test architectures using an actual shuttling compiler to see how they perform on realistic tasks.

We train RLIonS on a Q-chip with a capacity of up to 50 qubits in the same fashion as RLIonS was trained on the X-chip. We also train on a modified Q-chip, increasing the capacity of the SPAM zone from one to three qubits. As explained in Sec. II A 2, we expect that the duration of one shuttling step is shorter in this architecture when empty spots cross the junction. The actual time difference has to be determined experimentally after the chip is fabricated. Because the SAT solver only supports uniform step sizes, it is not applicable here. A heuristic compiler does not currently exist for this architecture.

We report RLIonS' performance on quantum volume circuits in Fig. 5. Ion shuttling on the standard Q-chip is consistently less efficient than on the X-chip: For up to 40 ions, it incurs roughly one extra shuttling step per two-qubit gate, and the overhead grows sharply beyond that. This abrupt rise can be explained by the storage ring reaching capacity, limiting opportunities to insert ions from other registers.

When raising the capacity of the SPAM register from one to three ions, the Q-chip performs almost as efficiently as the X-chip. This increase in intermediate storage options, which is likely achievable with no significant hardware changes, would therefore be a helpful mitigation design choice.

This analysis shows that our RL method is highly versatile and can successfully optimize ion shuttling across different architectures. We also demonstrate that it can be used to explore different hardware decisions during the design phase.

E. Ablations: analysis of RL design choices

Our proposed method contains various design decisions. In this section, we briefly evaluate the importance of some key components. The following ablations are considered: (1) Using a linear encoding of numeric values in the rep-

TABLE III. Ablation study on the importance of individual components of RLIonS. While all proposed design choices are important, our proposed representation has the largest effect. When using a more naive representation that is not invariant to qubit relabeling or commutative circuit reordering, the RL approach fails to solve even the smallest circuits.

Method	Largest solved quantum volume circuit
RLIonS	50 qubits
RLIonS (linear numeric encoding)	50 qubits
RLIonS ($\gamma = \gamma_s$)	26 qubits
RLIonS (no shaped reward)	26 qubits
RLIonS (naive representation)	0 qubits

resentation instead of the sinusoidal encoding described in Appendix C. (2) Setting $\gamma = \gamma_s$, such that the shaped reward from Sec. III E 1 does not introduce greediness. (3) Disabling the shaped reward completely. (4) Selecting a more naive representation (see Appendix D) than our proposed representation from Fig. 2 and Sec. III C 2.

An agent is trained for each ablation on the X-chip with a capacity of 50 qubits and is compared to the unmodified proposed method on quantum volume circuits. In Table III, the largest size of quantum volume circuits that can be solved is reported. A QV size is considered solvable if the agent reliably finds a valid shuttling sequence for 100 out of 100 test cases. For most ablations, the largest solvable size is severely reduced. A greedy-shaped reward is important, as without greediness, only up to 26 qubits can be solved. Without a shaped reward at all, again only 26 qubits can be solved. Our proposed representation has the largest influence by far. With a naive representation, the RL approach fails to compile any QV circuits, as it fails to learn the underlying patterns of the representation. When using a linear numeric encoding instead of the sinusoidal encoding, compilation remains reliable. We therefore briefly compare the quality of the compilations of both methods in Fig. 6. The agent using the linear encoding produces longer shuttling sequences for the same problems.

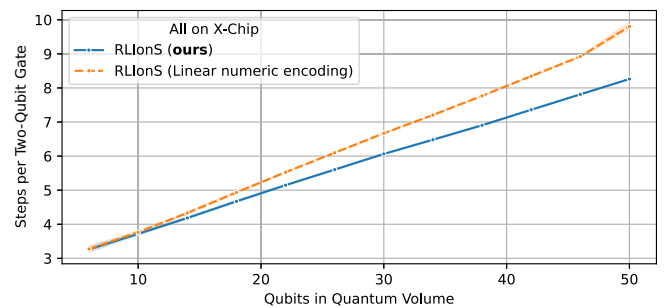


FIG. 6. Detailed study on the influence of numeric encodings in RLIonS, which is the only ablation experiment where compilation remains reliable up to 50 qubits. When using a linear encoding instead of the proposed sinusoidal encoding, the quality of shuttling sequences is reduced for problems larger than 10 qubits.

V. CONCLUSION AND OUTLOOK

We show that RL can produce efficient ion-shuttling schedules for trapped-ion quantum computers based on the QCCD architecture. Our schedules need up to 36.3% fewer shuttling operations than state-of-the-art heuristic compilers. As shuttling dominates circuit run-time, the speed-up substantially enhances the reliability of trapped-ion quantum computers since they are constrained by finite coherence times. For small instances, we benchmark our method, RLionS, against exact SAT solutions. In most cases, RLionS performs near optimally. For larger problems, exact solutions are intractable.

We also showcase the versatility of our approach, successfully applying it across different architectures. This makes our method useful during hardware development to assess how design choices affect shuttling efficiency.

Future work includes extending to more complex hardware, e.g., multijunction chips. As complexity increases, developing efficient heuristic strategies will become even more difficult. Our observation encoding could be generalized to further reduce shuttling overhead for circuits with long chains of commuting two-qubit gates. Furthermore, the initial qubit-to-ion mapping could be optimized using algorithms such as simulated annealing, evolutionary algorithms, or RL. Here, the duration of rollouts by a trained RLionS policy, or the faster estimate of a trained RLionS value function could be used as the objective.

Further investigation could focus on optimizing and evaluating design choices, such as ion swapping and chain splitting, and on minimizing shuttling sequences in architectures that support parallelism (for example, by performing multiple transports in storage registers while a gate is being executed). A possible implementation would require modifying the environment, adding joint and wait actions, and masking the action space to reflect whether a zone is busy, for example, during gate execution.

Another direction is to jointly optimize transpilation, guided by machine learning [48,49], with our shuttling optimization, as in a bilevel control problem, potentially further lowering the overall operation time.

Overall, our results underscore the importance of integrating methods from physics and computer science to advance trapped-ion quantum computing.

ACKNOWLEDGMENTS

We thank Celeste Torkzaban and Florian Ungerechts for the invaluable exchange on the current and planned chip designs at QVLS. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC-2123/2 QuantumFrontiers—390837967, the state of Lower Saxony and the VW Foundation through Quantum Valley Lower Saxony Q1 (QVLS-Q1), and the Federal Ministry of Research, Technology and Space (BMFTR) within the Quantum Computing Service Center QUICS (Grant No. 13N17418) and QuSTAC (Grant No. 13N17319). The publication of this article was funded by the Open Access Fund of the Leibniz Universität Hannover.

DATA AVAILABILITY

The source code and data that support the findings of this study are publicly available [50].

APPENDIX A: HYPERPARAMETERS

See Table IV for a list of the used values of all hyperparameters. These hyperparameters were determined empirically, with the exception of the step durations for the Q-chip. Here, the fast rotation speedup is an estimated value. The final value on the real hardware is currently not known. The lookahead depth is a trade-off between both inference and training time on one side and the quality of the shuttling sequence on the other side. The selected value of $k_{\text{lookahead}} = 4$ empirically showed good performance; for higher values, the improvements diminished.

APPENDIX B: HARDWARE REQUIREMENTS AND LIBRARIES

The SAT solver uses the Z3-solver [51] running on an Intel(R) Xeon(R) Gold 6242R CPU @ 3.10 GHz. The proposed RL approach is implemented using the JAX high-performance numerical computing library, running on an Intel(R) Core(TM) i9-9900K CPU @ 3.60 GHz and an NVIDIA GeForce RTX 2080 Ti. Training an agent with the hyperparameters from Table IV requires about 125 h on this configuration.

TABLE IV. Hyperparameters used for the PPO algorithm and the ResNet architecture, our proposed representation, and the Q-chip.

Hyperparameter	Value
Initial learning rate α	2.5×10^{-4}
Learning rate decay	Linear
Value function coefficient	0.5
Clipping factor ϵ	0.1
Parallel environments n_{envs}	250
Steps per environment n_{steps}	40
Entropy coefficient	1.0×10^{-4}
Maximum epochs per learning step	4
Mini batch size	1024
GAE λ	0.96
Discount factor $\gamma (=e^{-\beta})$	0.9995
Shaped reward discount factor γ_s	1.0
Penalty rate c_r	0.1
Training gate count n_{gates}	1275
Residual network blocks n_{blocks}	3
Hidden layer width n_{hidden}	512
Shared value/policy network	No
PPO learning steps	1000 000
Gate depth in representation $k_{\text{lookahead}}$	4
Sinusoidal bands cell numbers b_{cell}	6
Sinusoidal bands remaining total gates $b_{\text{total_gates}}$	7
X-chip movement duration	1.00 steps
Q-chip default movement duration	1.00 steps
Q-chip fast rotation duration	0.25 steps

APPENDIX C: SINUSOIDAL NUMERIC EMBEDDINGS

We employ a sinusoidal embedding [43] to encode the numeric values of the representation, such as cell numbers. This embedding is a parametrized function $S(x; x_{\max}, b)$ with the number of bands b and the maximum encoded value $x_{\max}$. These parameters are not optimized through gradient descent. S is defined as

$$S(x; x_{\max}, b) = \begin{pmatrix} x_{\text{norm}} \\ \cos(x_{\text{norm}} \cdot \pi \cdot 2^0) \\ \vdots \\ \cos(x_{\text{norm}} \cdot \pi \cdot 2^{b-1}) \\ \sin(x_{\text{norm}} \cdot \pi \cdot 2^0) \\ \vdots \\ \sin(x_{\text{norm}} \cdot \pi \cdot 2^{b-1}) \end{pmatrix}, \quad (\text{C1})$$

$$\text{where } x_{\text{norm}} = \text{clip}\left(\frac{x}{x_{\max}}, 0, 1\right). \quad (\text{C2})$$

A special case arises if x does not represent a valid number, i.e., when encoding the value $\diamond$ signaling empty. In this case, $S(\diamond; x_{\max}, b) = \mathbf{0}$.

APPENDIX D: NAIVE REPRESENTATION

In Sec. IV E, we use a more naive representation to highlight the importance of our proposed representation. The naive representation encodes the qubit positions as a vector of cells $\mathbf{v}_q$, where each entry is either the qubit label or zero if empty. For the example in Fig. 2, this would be $\mathbf{v}_q = (4, 1, 0, 0, 3, 5, 0, 0, 0, 2, 0)$. This is concatenated with a vector $\mathbf{v}_g$ describing the circuit by listing the remaining gates directly through the operand labels. This is padded with zeros to a fixed size. The circuit in Fig. 2 would be $\mathbf{v}_g = (1, 3, 2, 4, 1, 5, 1, 3, 0, 0, \dots)$. Importantly, this representation contains at least the same amount of information as our proposed representation, but does not alias equivalent states.

-
- [1] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, Trapped-ion quantum computing: Progress and challenges, *Appl. Phys. Rev.* **6**, 021314 (2019).
 - [2] J. I. Cirac and P. Zoller, Quantum computations with cold trapped ions, *Phys. Rev. Lett.* **74**, 4091 (1995).
 - [3] A. Sørensen and K. Mølmer, Quantum computation with ions in thermal motion, *Phys. Rev. Lett.* **82**, 1971 (1999).
 - [4] G. Zarantonello, H. Hahn, J. Morgner, M. Schulte, A. Bautista-Salvador, R. F. Werner, K. Hammerer, and C. Ospelkaus, Robust and resource-efficient microwave near-field entangling $^9\text{Be}^+$ gate, *Phys. Rev. Lett.* **123**, 260503 (2019).
 - [5] D. Kielpinski, C. Monroe, and D. J. Wineland, Architecture for a large-scale ion-trap quantum computer, *Nature (London)* **417**, 709 (2002).
 - [6] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, C. Ryan-Anderson, and B. Neyenhuis, Demonstration of the trapped-ion quantum CCD computer architecture, *Nature (London)* **592**, 209 (2021).
 - [7] S. A. Moses, C. H. Baldwin, M. S. Allman, R. Ancona, L. Ascarrunz, C. Barnes, J. Bartolotta, B. Bjork, P. Blanchard, M. Bohn, *et al.*, A race-track trapped-ion quantum processor, *Phys. Rev. X* **13**, 041052 (2023).
 - [8] J. Durandau, J. Wagner, F. Mailhot, C.-A. Brunet, F. Schmidt-Kaler, U. Poschinger, and Y. Bérubé-Lauzière, Automated generation of shuttling sequences for a linear segmented ion trap quantum computer, *Quantum* **7**, 1175 (2023).
 - [9] A. Ransford, M. S. Allman, J. Arkinstall, J. P. Campora III, S. F. Cooper, R. D. Delaney, J. M. Dreiling, B. Estey, C. Figgatt, A. Hall, *et al.*, Helios: A 98-qubit trapped-ion quantum computer, *arXiv:2511.05465*.
 - [10] T. Schmale, B. Temesi, A. Baishya, N. Pulido-Mateo, L. Krinner, T. Dubielzig, C. Ospelkaus, H. Weimer, and D. Borcharding, Backend compiler phases for trapped-ion quantum computers, in *2022 IEEE International Conference on Quantum Software (QSW)* (IEEE, Barcelona, Spain, 2022), pp. 32–37.
 - [11] A. A. Saki, R. O. Topaloglu, and S. Ghosh, Muzzle the shuttle: Efficient compilation for multi-trap trapped-ion quantum computers, in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, Antwerp, Belgium, 2022), pp. 322–327.
 - [12] P. Murali, D. M. Debroy, K. R. Brown, and M. Martonosi, Architecting noisy intermediate-scale trapped ion quantum computers, in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (IEEE, Valencia, Spain, 2020), pp. 529–542.
 - [13] X. Wu, C. Zhu, J. Wang, and X. Wang, MUSS-TI: Multi-level shuttle scheduling for large-scale entanglement module linked trapped-ion, in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture* (Association for Computing Machinery, New York, NY, USA, 2025), pp. 749–763.
 - [14] D. Schoenberger, S. Hillmich, M. Brandl, and R. Wille, Shuttling for scalable trapped-ion quantum computers, *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **44**, 2144 (2025).
 - [15] W. Dai, K. A. Brown, and T. G. Robertazzi, Advanced shuttle strategies for parallel QCCD architectures, *IEEE Trans. Quantum Eng.* **5**, 1 (2024).
 - [16] Quantum Valley Lower Saxony e. V., 2026, accessed: 20 May 2026, <https://qvl.de/en/>.
 - [17] R. Wille, L. Berent, T. Forster, J. Kunasaikaran, K. Mato, T. Peham, N. Quetschlich, D. Rovara, A. Sander, L. Schmid, D. Schoenberger, Y. Stade, and L. Burgholzer, The MQT handbook: A summary of design automation tools and software for quantum computing, in *IEEE International Conference on Quantum Software (QSW)* (IEEE, Shenzhen, China, 2024), pp. 1–8.
 - [18] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, Validating quantum computers using randomized model circuits, *Phys. Rev. A* **100**, 032328 (2019).

- [19] Quantum Valley Lower Saxony Q1–Forschung, 2026, accessed: 20 May 2026, www.qvls-q1.de/forschung.
- [20] D. Schoenberger, S. Hillmich, M. Brandl, and R. Wille, Using Boolean satisfiability for exact shuttling in trapped-ion quantum computers, in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, Incheon, Republic of Korea, 2024), pp. 127–133.
- [21] R. B. Blakestad, Transport of trapped-ion qubits within a scalable quantum processor, Ph.D. thesis, University of Colorado, 2010.
- [22] D. Hucul, M. Yeo, S. Olmschenk, C. Monroe, W. K. Hensinger and J. Rabchuk, On the transport of atomic ions in linear and multidimensional ion trap arrays, *Quantum Info. Comput.* **8**, 501 (2008).
- [23] F. Ungerechts, R. Munoz, A. Hoffmann, J. Bätge, M. M. Billah, T. Meiners, B. Kaune, G. Zarantonello, and C. Ospelkaus, Designing a trapped-ion quantum processor based on near-field microwave quantum logic gates (unpublished).
- [24] F. Ungerechts, J. Bätge, M. M. Billah, L. Krieger, R. Munoz, P. Nuschke, A. Hoffmann, G. Zarantonello, and C. Ospelkaus, CIRQL: A comprehensive register-based trapped-ion quantum processor with near-field microwave control (unpublished).
- [25] R. Bowler, J. Gaebler, Y. Lin, T. R. Tan, D. Hanneke, J. D. Jost, J. P. Home, D. Leibfried, and D. J. Wineland, Coherent diabatic ion transport and separation in a multizone trap array, *Phys. Rev. Lett.* **109**, 080502 (2012).
- [26] A. Walther, F. Ziesel, T. Ruster, S. T. Dawkins, K. Ott, M. Hettrich, K. Singer, F. Schmidt-Kaler, and U. Poschinger, Controlling fast transport of cold trapped ions, *Phys. Rev. Lett.* **109**, 080501 (2012).
- [27] X.-J. Lu, A. Ruschhaupt, and J. G. Muga, Fast shuttling of a particle under weak spring-constant noise of the moving trap, *Phys. Rev. A* **97**, 053402 (2018).
- [28] V. Kaushal, B. Lekitsch, A. Stahl, J. Hilder, D. Pijn, C. Schmiegelow, A. Bermudez, M. Müller, F. Schmidt-Kaler, and U. Poschinger, Shuttling-based trapped-ion quantum information processing, *AVS Quantum Sci.* **2**, 014101 (2020).
- [29] D. Schoenberger and R. Wille, Orchestrating multi-zone shuttling in trapped-ion quantum computers, in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Albuquerque, NM, USA, 2025), Vol. 1, p. 1069.
- [30] D. Schoenberger, J. Hilder, F. Schmidt-Kaler, and R. Wille, Shuttling for trapped-ion quantum computers with embedded processing zones, in *2025 IEEE International Conference on Quantum Software (QSW)* (IEEE, Helsinki, Finland, 2025), pp. 123–129.
- [31] T. Schmale, Hybrid quantum-classical computation—From infrastructure to algorithms, Institutionelles Repositorium der Leibniz Universität Hannover, Dissertation, 2026.
- [32] M. Sipser, Introduction to the theory of computation, *SIGACT News* **27**, 27 (1996).
- [33] X. Zhang and G.-Y. Zhu, A literature review of reinforcement learning methods applied to job-shop scheduling problems, *Comput. Oper. Res.* **175**, 106929 (2025).
- [34] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, Reinforcement learning for solving the vehicle routing problem, in *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS 2018)* (Neural Information Processing Systems Foundation, Inc., Montréal, Canada, 2018), Vol. 31.
- [35] C. Cummins, B. Wasti, J. Guo, B. Cui, J. Ansel, S. Gomez, S. Jain, J. Liu, O. Teytaud, B. Steiner, *et al.*, Compilergym: Robust, performant compiler optimization environments for AI research, in *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (IEEE, Seoul, Republic of Korea, 2022), pp. 92–105.
- [36] M. Bukov and F. Marquardt, Reinforcement learning for quantum technology, [arXiv:2601.18953](https://arxiv.org/abs/2601.18953).
- [37] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (John Wiley & Sons, Hoboken, NJ, USA, 2014).
- [38] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., edited by F. Bach (MIT Press, Cambridge, MA, USA, 2018), pp. 47–71.
- [39] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in *Proceedings of the 12th International Conference on Advances in Neural Information Processing Systems (NeurIPS 1999)* (MIT Press, Cambridge, MA, USA, 1999), Vol. 12.
- [40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, [arXiv:1707.06347](https://arxiv.org/abs/1707.06347).
- [41] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, High-dimensional continuous control using generalized advantage estimation, in *4th International Conference on Learning Representations, ICLR 2016* (OpenReview.net, San Juan, Puerto Rico, 2016).
- [42] R. Givan, T. Dean, and M. Greig, Equivalence notions and model minimization in Markov decision processes, *Artif. Intell.* **147**, 163 (2003).
- [43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, Attention is all you need, in *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS 2017)* (Curran Associates, Incorporated, Red Hook, NY, USA, 2017), Vol. 30.
- [44] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures, in *International Conference on Machine Learning* (PMLR, Stockholm, Sweden, 2018), pp. 1407–1416.
- [45] H. Lee, D. Hwang, D. Kim, H. Kim, J. J. Tai, K. Subramanian, P. R. Wurman, J. Choo, P. Stone, and T. Seno, Simba: Simplicity bias for scaling up parameters in deep reinforcement learning, in *13th International Conference on Learning Representations, ICLR 2025* (International Conference on Learning Representations, Singapore, 2025), pp. 50050–50082.
- [46] A. Y. Ng, D. Harada, and S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in *16th International Conference on Machine Learning* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999), Vol. 99, p. 278.

- [47] See Supplemental Material at <https://link.aps.org/supplemental/10.1103/b7ck-8wh4> for all results on MQT circuits and animations of some shuttling sequences.
- [48] B. Rosenhahn and T. J. Osborne, Monte Carlo graph search for quantum circuit optimization, *Phys. Rev. A* **108**, 062615 (2023).
- [49] B. Rosenhahn, T. J. Osborne, and C. Hirche, Optimization driven quantum circuit reduction, *New J. Phys.* **27**, 104509 (2025).
- [50] M. Schier and L. Richtmann, RLionS: Reinforcement learning for ion shuttling, GitHub, 2026, <https://github.com/m-schier/rlions/tree/v1.0>.
- [51] L. De Moura and N. Bjørner, Z3: An efficient SMT solver, in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Springer, Berlin, Heidelberg, Germany, 2008), pp. 337–340.