

# Quantum circuits as a game: A reinforcement learning agent for quantum compilation and its application to reconfigurable neutral atom arrays

Kouhei Nakaji,<sup>1,2</sup> Jonathan Wurtz,<sup>3</sup> Haozhe Huang\*,<sup>4,5</sup> Luis Mantilla Calderón\*,<sup>4,5</sup> Karthik Panicker,<sup>2</sup> Elica Kyoseva,<sup>1</sup> and Alán Aspuru-Guzik<sup>1,2,4,5,6</sup>

<sup>1</sup>*NVIDIA Corporation, 2788 San Tomas Expressway, Santa Clara, 95051, CA, USA*

<sup>2</sup>*Department of Chemistry, University of Toronto, Lash Miller Chemical Laboratories, 80 St. George Street, Toronto, ON M5S 3H6, Canada*

<sup>3</sup>*QuEra Computing Inc., 1284 Soldiers Field Road, Boston, MA, 02135, USA*

<sup>4</sup>*Department of Computer Science, University of Toronto,*

*Sandford Fleming Building, 10 King's College Road, Toronto, ON M5S 3G4, Canada*

<sup>5</sup>*Vector Institute for Artificial Intelligence, 661 University Ave. Suite 710, Toronto, ON M5G 1M1, Canada*

<sup>6</sup>*Acceleration Consortium, 700 University Ave., Toronto, ON M7A 2S4, Canada*

(Dated: June 12, 2025)

We introduce the “quantum circuit daemon” (QC-Daemon), a reinforcement learning agent for compiling quantum device operations aimed at efficient quantum hardware execution. We apply QC-Daemon to the move synthesis problem called the Atom Game, which involves orchestrating parallel circuits on reconfigurable neutral atom arrays. In our numerical simulation, the QC-Daemon is implemented by two different types of transformers with a physically motivated architecture and trained by a reinforcement learning algorithm. We observe a reduction of the logarithmic infidelity for various benchmark problems up to 100 qubits by intelligently changing the layout of atoms. Additionally, we demonstrate the transferability of our approach: a Transformer-based QC-Daemon trained on a diverse set of circuits successfully generalizes its learned strategy to previously unseen circuits.

## I. INTRODUCTION

Automated planning, often discussed in the context of artificial intelligence, robotics, and operations research, has garnered significant interest due to its ability to produce efficient plans or policies that achieve predefined objectives [1–4]. This approach is used to determine a sequence of actions—often constrained by available resources, temporal relationships, or environmental conditions—that leads from an initial state to a desired goal state. Techniques from reinforcement learning (RL) [5] and deep learning, particularly deep RL [6, 7], play a key role in automated planning.

The aim of this paper is to introduce automated planning of quantum device operations for executing quantum circuits. This planning, whose goal is to maximize the fidelity—i.e., the overlap between the ideal and actual output states—of an executed quantum circuit, is crucial for the efficient quantum circuit execution. During the execution of a circuit, the state of a device  $s_t$  changes as time evolves. This state can be the physical arrangement of qubits, the error rates for each qubit, the temperature of the quantum processor, or any other classical variable we can control. Inspired by the gaming testbeds for RL algorithms [8], we call the task of finding the optimal sequence of actions that maximize the fidelity of a quantum circuit by controlling the variables in  $s_t$  the *quantum circuit game* (QC-Game). We formalize such planning game as a Markov decision process, and propose an AI solver that we call the quantum circuit daemon (QC-Daemon).

The recent development of quantum devices highlights the need to effectively plan and synthesize the execution of quantum device operations. For example, quantum hardware based on reconfigurable atom arrays (RAA) [9, 10] has the freedom to change the layout of atoms, each storing a qubit, during computation. Hence, we can consider  $s_t$  to be the layout and plan its transition accordingly. The recent advances of logical quantum processors [10–15] also emphasizes the importance of careful planning of resource preparation for fault-tolerant quantum computation.

We focus on the application of automated planning to RAA, with a particular emphasis on the important role of RL in enhancing planning capabilities. To this end, we build instances of both the QC-Game and QC-Daemon tailored to this type of quantum processor. We design a QC-Game environment, termed the *Atom Game*, where the QC-Daemon learns to dynamically reconfigure atom layouts while executing quantum gates. Our implementation employs a Transformer-based model having a physically motivated architecture for the QC-Daemon, enabling it to flexibly handle varying circuit structures and atom configurations. Reinforcement learning plays a crucial role in maximizing the benefits of automated planning: the RL-trained QC-Daemon achieves a considerable reduction in the total cost represented as the logarithmic infidelity across benchmark experiments involving circuits with up to 100 qubits. Importantly, we demonstrate the transferability of our approach by showing that a Transformer-based QC-Daemon trained on a diverse set of circuits can successfully apply its learned strategy to previously unseen circuits.

This paper contributes to the growing subfield of RAA

\* These authors contributed equally.

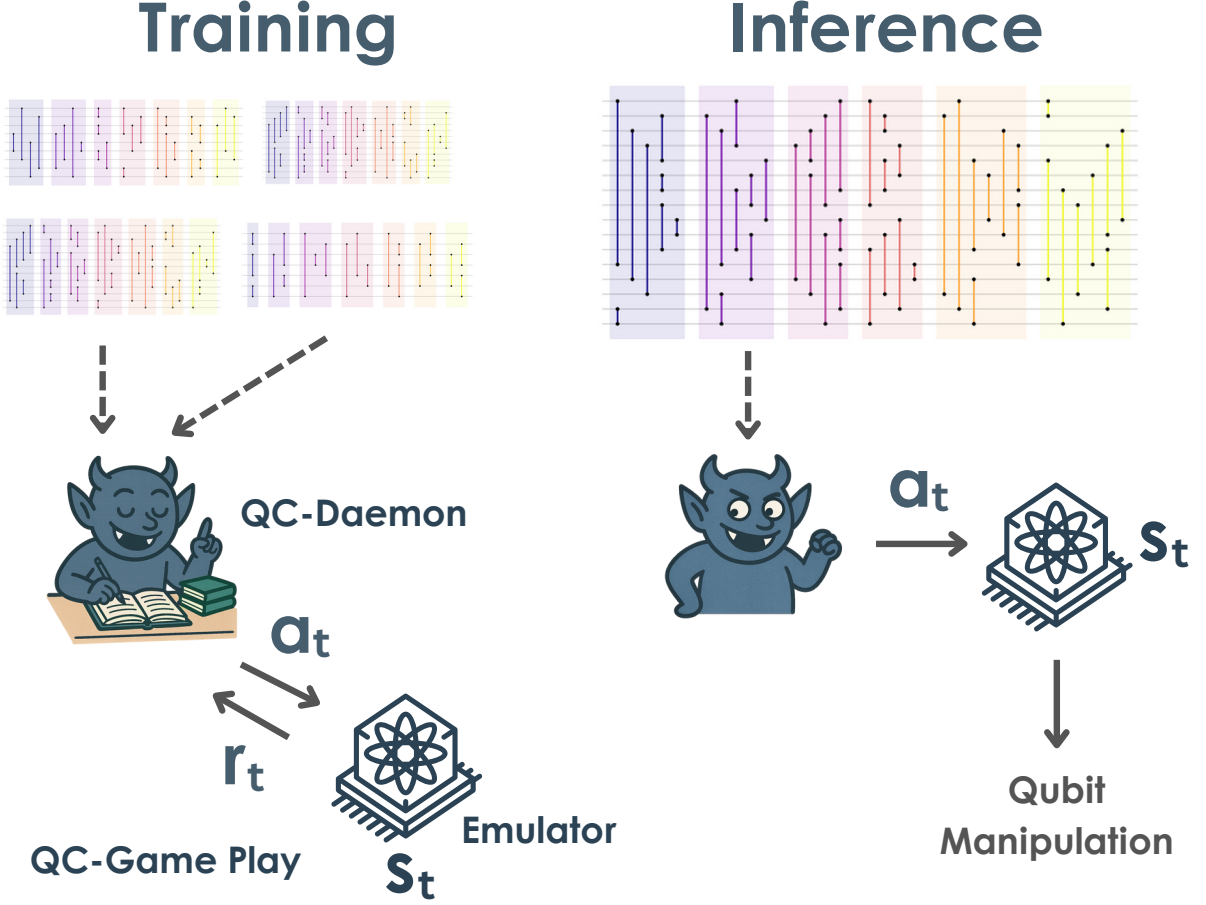


FIG. 1. Training and inference with the QC-Game. During training, QC-Daemon accumulates experiences from playing the QC-Game with quantum circuits, gets feedback  $r_t$ , and updates its parameters. During inference, it plays the QC-Game to generate qubit manipulations to execute the target circuit based on its experiences.

*move synthesis*, which maps abstract circuit executions to hardware-level instructions of RAAs. Due to their inherent flexibility, the mapping generalizes the layout synthesis problem [16] on a superconducting architecture. The novel contribution of this paper is to consider the move synthesis problem from an RL perspective [17]. Our work also contributes to the field of AI for quantum computing [18] by proposing a physically motivated architecture suitable for quantum computation.

It should be noted that even though we focus on optimizing the reconfiguration step to simplify the problem structure for the first AI-based neutral atom compiler, the other processes may become more important depending on the problem. For example, if the circuit has a sparse and repetitive structure, optimizing the initial layout may be more important than the reconfiguration step. Our RL compiler can be extended to handle other operations, including initialization, which we leave for future work.

The rest of the paper is organized as follows. In Section II, we introduce the QC-Game and QC-Daemon. In

Section III, we describe the Atom Game, an instance of the QC-Game for neutral atom arrays. We develop a QC-Daemon to solve the Atom Game in Section IV. Section V is dedicated to study the effects of the dynamic layout reconfiguration using the proposed solvers. We conclude with some discussions in Section VI.

## II. THE GAME OF QUANTUM CIRCUITS

In this section, we describe a general framework for building the QC-Daemon, an automated planning agent for quantum circuits execution. First, we design the QC-Game for optimizing the sequence of device operations in Section II A and then introduce its player, the QC-Daemon, in Section II B.

### A. The QC-Game

The QC-Game consists of finding the optimal sequence of device operations to execute a given quantum circuit while controlling the variables that affect its execution. This game can be formulated with a Markov decision processes  $(\mathcal{S}, \mathcal{A}, P, R)$  as we explain below.

Given some circuit  $\mathcal{C}$ , a pre-processing step decomposes the circuit into a sequence of parallelizable sets of one-qubit gates, and parallelizable sets of two-qubit gates [19]. This decomposition can be done in many ways, such as an ASAP heuristic [20]. Then, we ignore all one-qubit gates for purposes of move synthesis since they can be performed locally and with much higher fidelity compared to the two-qubit gates. The resulting data is  $T$  disjoint chunks representing parallel two qubit gate executions  $\mathcal{C} = \{C_t\}_{t=0}^{T-1}$ , where all gates in  $C_t$  commute and act on independent qubits. Consequently, each chunk can be implemented in one time step. This work presumes  $\mathcal{C}$  is fixed; future work may co-optimize the circuit parallelism consecutively with the move synthesis step.

Due to hardware constraints, implementing  $C_t$  requires many actions, such as moving atoms around in the array to obey gate adjacency constraints. To this end, the player of the game then plans actions for each chunk  $C_t$  given a device configuration  $s_t \in \mathcal{S}$  and all future chunks  $\mathcal{C}_{t:T} := \{C_j\}_{j=t}^{T-1}$ . Each action  $a_t \in \mathcal{A}_t$  is chosen from the set of feasible device operations at time  $t$ . Each action space  $\mathcal{A}_t$  is a subset of  $\mathcal{A}$ , the set of all possible device operations. The game then evolves as  $s_{t+1} \sim P(s_t, a_t, C_t)$ , where  $P : \mathcal{S} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{P}(\mathcal{S})$  is the transition kernel—which depends on the underlying quantum device, and ends when  $t$  becomes  $T$ . For simplicity, we let  $P$  be a deterministic function  $P : \mathcal{S} \times \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{S}$  in our simulations, but in real settings, this assumption might not hold.

The goal of the QC-Game is to maximize the fidelity of implementing the target circuit. However, measuring this fidelity is inconvenient in practice and thus requires a proxy that is easy to compute and relates to the circuit fidelity. Examples of such proxy include the execution time or the gate depth of the circuit, since both tend to increase decoherence and decrease circuit fidelity. We can then define a reward model  $R : \mathcal{S} \times \mathcal{S} \times \mathcal{C} \rightarrow \mathbb{R}$ , based on the logarithmic infidelity, which counts actions weighted by the (log) fidelity of that action as a cost. Each action  $a_t$  thus has a signal  $r_t = R(s_t, s_{t+1}, C_t)$ . This allows us to rewrite the objective of QC-Game as finding the optimal sequence of actions that maximize the cumulative reward during the circuit execution

$$(a_t^*)_{t \geq 0} = \operatorname{argmax}_{\forall t: a_t} \sum_{t \geq 0} \gamma^t r_t, \quad (1)$$

where  $\gamma \in (0, 1]$  is a discount factor.

We emphasize that the QC-Game and what is commonly referred by *quantum circuit compilation* are different and achieve complementary goals. Given a target

circuit  $\mathcal{C}$ , quantum circuit compilation aims to reduce the depth of such circuit by creating a shorter but equivalent quantum circuit at the software level [21–23], and multiple RL algorithms have been proposed for such problem [24–26]. In contrast, the QC-Game aims to implement  $\mathcal{C}$  in the best possible way for a specific quantum processor to implement a given circuit. The QC-Game is closely related to *quantum layout synthesis* and both routines share the same goal. However, most layout synthesis algorithms only optimize device operations constrained to  $C_j$  for every  $j$  [27]—similar to an iterative greedy optimization

$$(a_t^*)_{t \geq 0} = \left( \operatorname{argmax}_{a_t} r_t \right)_{t \geq 0}, \quad (2)$$

whereas the QC-Game does so constrained to  $\mathcal{C}_{j:T}$ —leveraging global information.

### B. The QC-Daemon

Given the exponentially large state and action space of the QC-Game, we need to search an approximate solution. We focus on designing a policy function  $\pi$ , a well-studied technique in optimal control theory, which provides the probability distribution of actions conditioned to a given state.

For the QC-Game, we define the QC-Daemon to be a policy function

$$\pi(a_t | s_t, t, \mathcal{C}_{t:T}) \quad (3)$$

that is aware of the current time step  $t$ , the remaining quantum circuit  $\mathcal{C}_{t:T}$ , and the device state  $s_t$ . Formally, in RL, both  $t$  and  $\mathcal{C}_{t:T}$  are included in the definition of the state  $s_t$ , but we explicitly separate them to emphasize this look-ahead characteristic of the QC-Daemon.

The QC-Daemon can be implemented by using a neural network  $\pi_\theta$ , and in that case, each input is often converted to an embedding—a vector of  $k$  real components that, after training, aims to be a better representation of the input. We note that the function that creates an embedding of the upcoming circuit  $\mathcal{C}_{t:T}$ , which we call a *quantum circuit feature*, is reusable in different types of QC-Games. Studying the transferability of quantum circuit features across multiple QC-Game settings is an interesting direction for future research.

QC-Daemon uses QC-Game in two ways (Fig. 1): Training and inference with the QC-Game. During training, QC-Daemon accumulates experiences from playing the QC-Game with quantum circuits, gets feedback  $r_t$ , and updates its parameters. During inference, it plays the QC-Game to generate qubit manipulations to execute the target circuit based on its experiences.

### III. APPLICATIONS IN RECONFIGURABLE ATOM ARRAYS

In this section, we introduce the Atom Game, a specific instance of the QC-Game, designed for RAAs. The Atom Game addresses the problem of planning a sequence of layout reconfigurations, i.e. scheduling the 2D positions of each atom, for an input circuit (cf. Fig. 2). First, Section III A provides an overview of neutral atom arrays and Section III B explains in detail the Atom Game. Then, Section III C describes one proxy for circuit fidelity based on the noise introduced when moving atoms and executing gates. This proxy is closely related to the logarithmic infidelity of the circuit. Finally, Section III D briefly discusses an alternative application of the Atom Game to a neutral atom logical processor.

#### A. Reconfigurable neutral atom arrays

Reconfigurable atom arrays (RAAs) are a quantum computing architecture based on optically trapped neutral atoms, where each atom hosts a qubit in its electronic ground states [28]. Each neutral atom's position is fixed in optical traps using a spatial light modulator (SLM) laser. For simplicity, we define the two-dimensional positions of the traps as  $\mathcal{V} \subset \mathbb{N} \times \mathbb{N}$ . The device state is then the joint position of all  $N$  atoms

$$s_t := \left( \mathbf{v}_q^{(t)} \right)_{q=1}^N \in \mathcal{V}^N, \quad (4)$$

where  $\mathbf{v}_q^{(t)} := (c_q^{(t)}, r_q^{(t)}) \in \mathcal{V}$  denotes the position of the  $q$ -th atom at time  $t$  and  $N$  is the number of atoms.

The movement of atoms is implemented by means of dynamical tweezers created by a laser controlled by crossed acousto-optic deflectors (AOD). The AOD laser projects both horizontal and vertical beams, forming a 2D grid of optical traps, with one trap at each intersection of every row and column. These traps can capture, move, and release atoms by turning on and off and moving the positions of the rows and columns, enabling changes of the positions  $s_t$ .

Single-qubit operations are performed within each occupied trap by a Raman laser system [29, 30]. In contrast, two-qubit operations involve bringing two atoms into close proximity and exciting them into Rydberg states with a global Rydberg laser [9, 11]. In this work, we assume a “zoned” register layout with a storage and a gate region, as shown in Fig. 2. For each two-qubit parallel gate chunk  $C_t \in \mathcal{C}$ , atoms are moved from a storage region to the gate region using the dynamic tweezers. Then, a laser enacts a CZ gate between each pair of atoms in the gate region. Then, atoms are moved back to the storage region in preparation for the next set of gates.

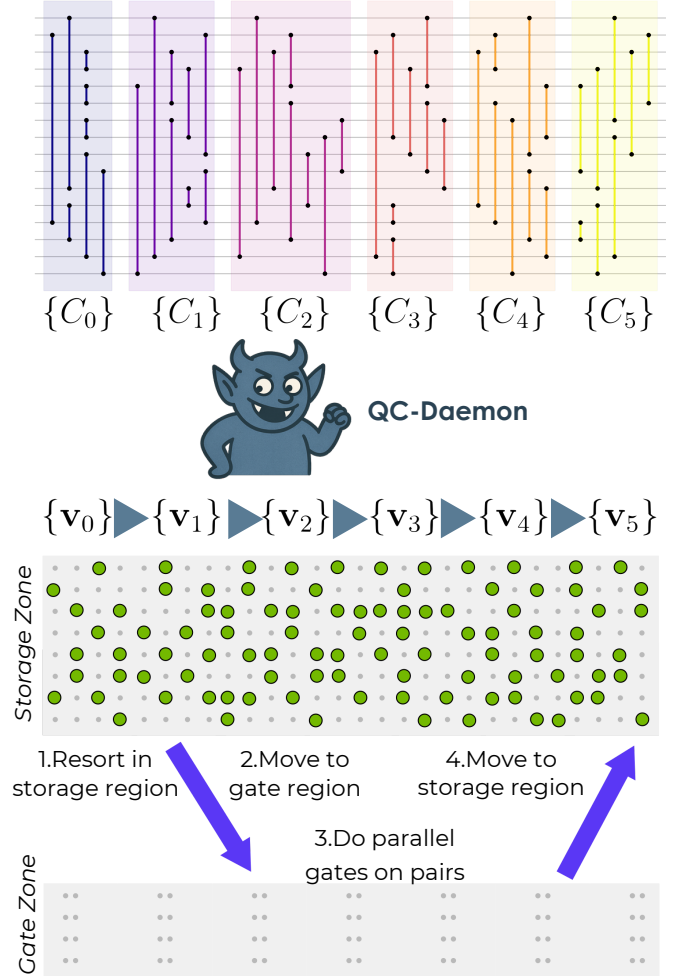


FIG. 2. The structure of the atom game on a zoned reconfigurable atom array. Given a sequence of parallel 2-qubit gate chunks  $\mathcal{C} = \{C_t\}_{t \geq 0}$ , the QC-Daemon synthesizes moves by defining atom positions  $s_t$  that minimize some objective given by the efficiency of implementing that reconfiguration. Then, a back-end compiler synthesizes those reconfigurations into atom moves with the AODs. The computation is executed in 4 steps: first, atoms are resorted in the storage region; then, atoms are moved to the gate region; then a parallel entangling CZ gate is done on all pairs; then, all atoms are moved back to the same positions, and the process repeats for all chunks.

#### B. The Atom Game

The Atom Game is a QC-Game where the goal is to choose the device state  $s_t$ , for each parallel gate chunk  $C_t$ , that minimizes the overhead cost of gate execution caused by atom reconfiguration. Having determined a state for each gate chunk, the complete circuit is executed by repeating four steps each time-step  $t$ , as represented in Fig. 2. First, atoms are loaded if the storage region is empty, or reconfigured otherwise, to state  $s_t$  and then all necessary one-qubit gates are performed in this region. Second, atoms participating in the parallel two-qubit gates of the corresponding time-step are moved



adjacent to each other in the gate region. Third, a parallel CZ entangling gate is done on all pairs present in the gate region. Finally, all atoms are moved back to their original locations in the storage region. This process is then repeated until the circuit is fully executed.

The problem of finding near-optimal laser control steps for a given layout of atoms  $s_t$  and two-qubit gates  $C_t$  has been studied in the previous literature [20, 31–37]. In most of these studies, the layouts before and after gate execution (i.e., the atom positions from steps 1 and 4 in Fig. 2) within a chunk remain the same. However, Ref. [35] introduces a method that modifies the layout after the gate execution to minimize the atoms’ moving distance to implement  $C_t$ . For ease of discussion, we assume the former case of having the same layout, but our approach can be easily extended to the latter case.

In most of these previous methods, we can calculate the cost, such as the circuit infidelity, associated with applying gates  $C_t$  in a given configuration of atoms  $s_j$ . We call the cost of such execution

$$G(s_t, C_t). \quad (5)$$

Similarly, for a given layout change  $a_t : s_t \mapsto s_{t+1}$ , we call its cost of movement

$$L(s_t, s_{t+1}), \quad (6)$$

and refer to  $L$  as the layout function. With these cost functions, we define the reward at time  $t$  as

$$R(s_t, s_{t+1}, C_t) := -L(s_t, s_{t+1}) - G(s_{t+1}, C_t) + G(s_0, C_t), \quad (7)$$

where the last term is the gate cost when using the initial configuration  $s_0$  and acts as a baseline reference.

Finally, to synthesize the actual reconfiguration actions, a lower-level compiler then converts each reconfiguration step into a sequence of valid AOD moves: first, perform  $a_t$  to reconfigure the positions of atoms in the storage region, and then move atoms to execute the two-qubit gates. The process of Atom Game is summarized in Algorithm 1.

---

**Algorithm 1** One run of the Atom Game

---

**Require:**  $s_0, \{C_t\}_{t=0}^{T-1}, \pi$   
1: **for**  $t \leftarrow 1$  to  $T$  **do**  
2:   Sample  $a_t \sim \pi(a_t | s_t, t, C_{t:T})$   
3:   Set  $s_{t+1} = P(s_t, a_t, C_t)$   
4:    $r_t = -L(s_t, s_{t+1}) - G(s_{t+1}, C_t) + G(s_0, C_t)$   
5: **end for**  
6: **return**  $\{s_t, a_t, r_t\}_{t \geq 0}$

---

Previous works have focused on minimizing  $G(s, C_t)$ . However, our study explores how to reconfigure the atoms for a given function  $G$  using the formalism of the QC-Game. Therefore, although we provide a specific protocol for estimating  $G$  and  $L$  in Section III C,

our approach applies to any definition.

### C. Reconfiguration cost estimation

To reconfigure the storage region from layout  $s_t$  to  $s_{t+1}$  or implement a set of gates  $C_t$  from layout  $s_t$ , the dynamic AODs must implement a sequence of moves satisfying constraints. For each chunk  $C_t$ , we can associate a cost—or, equivalently, a reward—that reflects the log infidelity of executing the underlying AOD moves:

$$\mathcal{J}(D, M) = \alpha DN + \beta M. \quad (8)$$

The first term represents idling error, where  $D$  is the duration of the moves,  $N$  is the total number of qubits in the array, and  $\alpha \geq 0$  is the inverse coherence time of each qubit. The second term represents the move error, where  $M$  is the number of qubits “touched”, e.g., participating in that move, and  $\beta \geq 0$  is a fixed cost corresponding to the fidelity loss from picking up and dropping off an atom to and from an AOD. By using the function  $\mathcal{J}$ , we model the cost for gate execution and the layout change as follows:

$$L(s_t, s_{t+1}) = \mathcal{J}(D_L(s_t, s_{t+1}), M_L(s_t, s_{t+1})), \quad (9)$$

$$G(s_t, C_t) = \mathcal{J}(D_G(s_t, C_t), M_G(s_t, C_t)), \quad (10)$$

where  $D_L(s_t, s_{t+1})$  and  $M_L(s_t, s_{t+1})$  are the duration and the number of touches for the layout change  $s_t \mapsto s_{t+1}$ , while  $D_G(s_t, C_t)$  and  $M_G(s_t, C_t)$  correspond to these values when executing the gates  $C_t$  from the layout  $s_t$ .

Instead of explicitly constructing a sequence of AOD moves to calculate a reconfiguration cost—as done in previous works [20, 31–37]—we instead estimate  $D_L, M_L, D_G$ , and  $M_G$  with an upper bound that depends on the number of moves, a value obtained with a graph-theoretic heuristic. The steps to estimate  $D_L(s_t, s_{t+1})$  and  $M_L(s_t, s_{t+1})$  are as follows:

#### 1. Accumulate the active participants

Given a layout change  $s_t \mapsto s_{t+1}$ , identify all atoms  $q$  for which  $\mathbf{v}_q^{(t)} \neq \mathbf{v}_q^{(t+1)}$ . From that subset, identify all “active” columns and rows

$$A_c = \bigcup_q \{c_q^{(t)}, c_q^{(t+1)}\}, \quad A_r = \bigcup_q \{r_q^{(t)}, r_q^{(t+1)}\}$$

in that rearrangement. The active participants  $A$  are the atoms that are in one of the active rows or columns and are consistent with the crossed AOD constraints. This set may include atoms which do not change location, as they get “caught” in the crossed AOD.

#### 2. Estimate the number of moves

Given the active participants  $A$ , the number of moves  $n_m^{(t)}$  is computed based on a log-depth reconfiguration heuristic [38] with infinite swap space.

For more details on estimating  $n_m^{(t)}$ , see Sec. III C 2.

### 3. Compute the total reconfiguration cost

The total number of touches is equal to the number of moves times the number of active participants  $M_L = \epsilon n_m^{(t)} |A|$  scaled by some constant  $\epsilon \sim 1$  that estimates how many atoms are touched per move. Here, we assume that half the atoms are moved per step  $\epsilon = 0.5$ . The total duration is equal to the number of moves times the characteristic time per move  $D_L = n_m^{(t)} \tau$ . Assuming constant acceleration moves [10], the characteristic time is the square root of the maximum distance between the start and end of all the active participants,

$$\tau = \max_{q \in A} \left[ \sqrt{|\mathbf{v}_q^{(t)} - \mathbf{v}_q^{(t+1)}| / \gamma} \right],$$

where  $\gamma$  is a constant having the dimension of the acceleration.

We approximate  $D_G(s_t, C_t)$  and  $M_G(s_t, C_t)$  with a two-step process. The first step moves the relevant pairs of atoms within the storage region in  $C_t$  to be adjacent. The second step then moves such pairs of atoms to the gate region in parallel to execute  $C_t$ . This results in

$$D_G(s_t, C_t) = D_L(s_t, \tilde{s}_t) + T_G, \quad (11)$$

$$M_G(s_t, C_t) = M_L(s_t, \tilde{s}_t), \quad (12)$$

where  $\tilde{s}_t$  is the intermediate state in which atoms are reconfigured to be adjacent in the storage region. The term  $T_G$  denotes the total time to move atoms from the storage to the gate region and back.

#### 1. The conflict graph

The *Conflict Graph*  $\mathcal{G}_c$  is a tool to schedule moves under the crossed AOD constraints and is defined as follows. Given a layout change  $s_t \mapsto s_{t+1}$ , each vertex is an atom participating in the change, with an edge between vertices if the two moves cannot be done in parallel due to violating the crossed AOD constraints (cf Fig. 3). There are two classes of edges in the conflict graph corresponding to the following two constraints:

- **Many-to-one constraint:** An AOD row cannot split into multiple rows, and multiple rows cannot merge into one. For example, in one time step

$$\begin{aligned} ((2, 0), (1, 2)) &\mapsto ((6, 0), (5, 1)) \\ ((2, 0), (1, 2)) &\not\mapsto ((6, 4), (5, 1)). \end{aligned}$$

The second move is invalid, as the AOD row picking up row 0 cannot split to drop onto rows 2 and 3 (Fig. 3 top left).

- **Ordering constraint:** Two rows or columns of the AOD may not cross each other during the move, so two atoms may not move out of order. For example, in

one time step

$$\begin{aligned} ((2, 0), (1, 2)) &\mapsto ((5, 1), (4, 4)) \\ ((2, 0), (1, 2)) &\not\mapsto ((5, 1), (5, 4)). \end{aligned}$$

The second move is invalid, as the AOD columns have crossed during the transfer (Fig. 3 bottom left). Code to compute  $D$  is provided here [39].

#### 2. Estimating the number of moves $n_m^{(t)}$

A simple parallel move scheduler may be implemented using a vertex coloring of the conflict graph  $\mathcal{G}_c$ . A vertex colouring partitions  $k$  subsets of vertices such that every partition is an independent set with no two vertices sharing an edge. If a subset of vertices on the conflict is an independent set, it can be done in a single AOD move, as no two pairs of moves conflict.

Thus, moves can be scheduled in any order by moving each partition in a single move from start to end. The number of moves equals  $k$ , and each atom is touched exactly once. Finding the minimum number of partitions  $k_{\min}$ , known as the chromatic number, is an archetypal NP-complete optimization problem. Brooks' theorem states that the chromatic number of a graph is upper bounded by the maximum connectivity of the graph [40], which can be found efficiently with a greedy heuristic. Thus, an (over)estimate of  $n_m^{(t)} = \Delta(\mathcal{G}_c) + 1$ , the maximum connectivity of the conflict graph.

A more advanced move scheduler can be implemented based on a divide-and-conquer approach first introduced in [38] by adding intermediate moves in a swap region to some intermediate positions  $s_t \rightarrow s' \rightarrow s'' \rightarrow \dots \rightarrow s_{t+1}$ . Crucially, the new conflict graph of each intermediate move is the union of the two induced subgraphs of the original conflict graph, plus extra edges from the many-to-one constraint. If an extensive number of edges can always be removed in this manner with a MaxCut heuristic, the number of moves required to remove all edges and thus reconfigure the layout is logarithmic in the number of edges in the conflict graph  $n_m^{(t)} = \delta \log(|V(\mathcal{G}_c)|)$ . This is a divide-and-conquer approach, as the conflict graph is iteratively partitioned into smaller and smaller graphs until the resorting is complete. It is beyond the scope of this work to analyze the specific action of this iterative partitioning. Here we simply assume that the swap region is large enough so that the landscape is convex and a greedy MaxCut heuristic always succeeds. The particular exponent of the partitioning is represented by  $\delta \sim 1$  which we assume to be order 1 in this work.

Thus, to estimate the number of moves, we count the number of edges in the conflict graph and take the log (in any base). The iteration exponent  $\delta$  is then an overall scale factor of the cost function  $n_m^{(t)} \rightarrow \delta n_m^{(t)}$ , which is irrelevant to the action of the QC-Daemon.

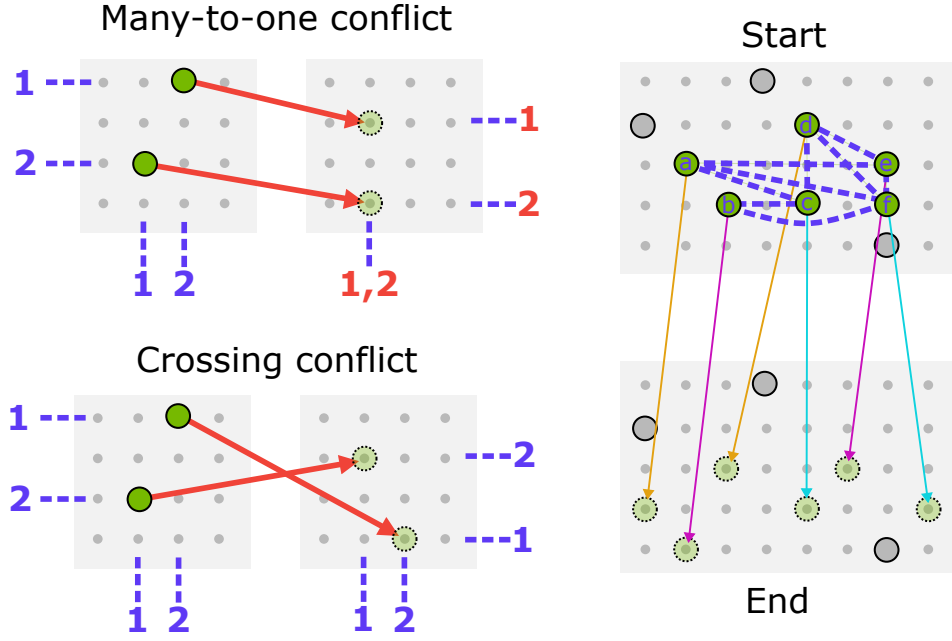


FIG. 3. **Defining the conflict graph.** Left: the two kinds of parallel moves disallowed by the crossed AOD constraints. Top: a many-to-one conflict, where two AOD rows/columns (purple) merge or separate into one row/column. Bottom: a crossing conflict, where two rows/columns change order during the move. This can happen in both the X and Y directions. Right: an example reconfiguration, with active participants in green and inactive participants in grey. The conflict graph edges are shown as purple dashed; it is three-colorable, as shown in the three colours of parallel directed moves.

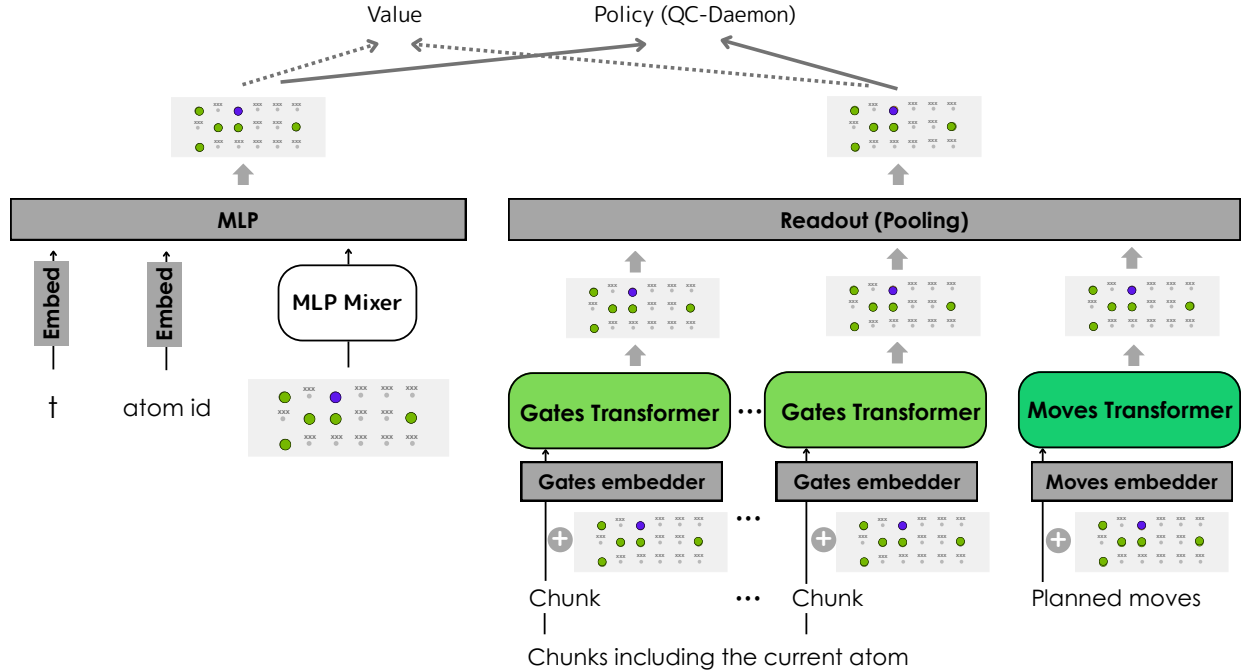


FIG. 4. The implementation of the atom-by-atom policy (QC-Daemon)  $\pi^A$  and the value function. Our model comprises two parts: static feature extraction (left) and dynamic feature extraction, which are implemented using the Gate Transformer and Move Transformer (right). The static features are independent of the circuit and the planned moves, and indicate which traps are more favorable for each atom at each time step. The dynamic features are conditioned on the circuit and the already planned moves.

#### D. Remark on logical processor

Even though we demonstrate the framework for moving individual physical qubits, we can also use the same Atom Game framework for moving groups of physical qubits in a coarse-grained manner. This opens the door to actively optimizing the layout of the qubits in an error-corrected quantum computer using reinforcement learning, as the constraints of such a system may be similar to those present in crossed AOD grids [38].

Using our compilation approach on groups of atoms, where each group corresponds to a particular logical qubit and is moved all together, can enable larger amounts of parallelization in the logical transversal gates of a circuit [10, 41]. Furthermore, abstracting away the hardware implementation, the Atom Game framework could be used for circuit compilation in topological codes, where logical qubits are “carved out” on a grid, as it happens with surface codes [42, 43]. In such cases, logical qubits can be defined via holes or patches in a grid, and logical operations are performed via braiding [44] or lattice surgery [45], respectively. Thus, each logical qubit is defined within a spatial region in the topological code grid that changes throughout the circuit implementation [46], and consequently, will have a cost associated to its movement with such a grid.

Previously, surface code compilation has been solved with heuristic algorithms [47–49]. These compilers optimize over the logical lattice instructions that allow implementing quantum algorithms, such as entangling measurements with lattice surgery merges and splits, distillation of magic states in specific regions, and other instructions that are required for universality. Given its resemblance to the RAA compilation problem, the QC-Game framework would similarly benefit circuit compilation at the logical circuit level, accounting for information encoded in the reward function that would otherwise be challenging to include in non-data-driven methods.

### IV. PLAYING THE ATOM GAME

In this section, we introduce our QC-Daemon for Atom Game. Our QC-Daemon determines the actions of each ‘playable atom’ for each time  $t$ . The playable atoms are chosen in the following manner. Let  $f$  be the function to extract the list of qubit indices included in  $C_t$ . The playable atoms  $P_t$  are defined as

$$P_t = \bigcup_{w=0}^{W-1} f(C_{t+w}), \quad (13)$$

where  $W$  (Window Size) is the hyperparameter. Namely, the atoms included in the gates in the next  $W$  chunks are used as the playable atoms.

#### A. Auto regressive action generation

The action generation is performed by determining the action of each playable atom in auto-regressive manner. Let  $P_t = (q_b)_{b=1}^{|P_t|}$ . The QC-Daemon policy can be decomposed as

$$\pi(a_t | s_t, t, C_{t:T}) = \prod_b \pi^A \left( a_t^{(b)} | s_t, t, C_{t:T}, q_b, \left( a_t^{(b')} \right)_{b'=1}^{b-1} \right), \quad (14)$$

where the function  $\pi^A$  is an atom-by-atom policy that determines the action for each playable atom  $q_b$ . Each action specifies only the next position of the atom. The action  $a_t^{(b)}$  generated for atom  $b$  is used in subsequent action generations for each playable atom. These individual actions are finally aggregated into the overall action as  $a_t = \left( a_t^{(b)} \right)_{b=1}^{|P_t|}$ .

#### B. Our model

Here, we describe our model for  $\pi^A$ . For notational convenience, we write  $s_t = (\mathbf{v}_q)_{q=1}^N$  and the layout after applying  $\left( a_t^{(b')} \right)_{b'=1}^{b-1}$  to  $s_t$  as  $(\mathbf{v}'_q)_{q=1}^N$  in this section. We also write the current playable atom as  $q_0$ .

Our model comprises two parts: static feature extraction and dynamic feature extraction. The static features do not depend on the circuit or the planned moves, and describe which traps are more favorable for each atom at each time step. The dynamic features are conditioned on the circuit and the already planned moves. The input of static features is the time  $t$ , the playable atom’s id, and  $(\mathbf{v}'_q)_{q=1}^N$ . In contrast, the input of the dynamic features encompass the subsequent chunks  $\mathbf{c} = C_{t_1}, \dots, C_{t_K}$  that include the current playable atom, where  $K$  (Horizon-Length) is a hyper-parameter, as well as  $(\mathbf{v}_q)_{q=1}^N$  and  $(\mathbf{v}'_q)_{q=1}^N$ .

##### 1. Static feature extraction

The input of the static feature extraction is converted into a vector embedding:  $t \rightarrow \mathbf{e}_t$ , atom id  $\rightarrow \mathbf{e}_a$ , and  $(\mathbf{v}'_q)_{q=1}^N \rightarrow \mathbf{e}_l$ . They are then concatenated into a vector  $\mathbf{e}$ :

$$\mathbf{e} = \mathbf{e}_t \oplus \mathbf{e}_a \oplus \mathbf{e}_l \quad (15)$$

Finally, a multilayer perceptron is applied to  $\mathbf{e}$  to obtain  $\mathbf{d} := \{d_j\}_{j=1}^{n_{\text{grid}}}$ .

For building  $\mathbf{e}_l$ , we use the MLP-Mixer [50]. The MLP-Mixer is a neural network architecture that relies solely on multi-layer perceptrons (MLPs), without using



convolutions or attention mechanisms. It operates on image patches rather than individual pixels, processing data through two types of MLPs: one mixes information across spatial (patch) dimensions and another mixes across channel dimensions. This separation allows the MLP-Mixer to capture local and global data patterns. In our case, we use  $(\mathbf{v}_q)_{q=1}^N$  as the input to the MLP-Mixer, treating each spatial cell as a patch embedding that contains one-hot encoded atom assignments.

## 2. Dynamic feature extraction

Two types of Transformers, the Gate Transformer and the Move Transformer, are used to extract dynamic features. The output from those components is combined and processed in the readout layer.

*a. Gates Transformer* In each chunk, we consider the gate containing the current playable atom (PA-gate) along with other gates (non-PA-gates) included in the chunk. The Transformer layer captures correlations between these non-PA-gates and the PA-gate by assuming that the playable atom moves to each position of the grids.

The input to the Transformer layer is constructed via a gate embedder, whose output is  $E_{\text{atoms}}^g \oplus E_{\text{grids}}^g$ , where  $E_{\text{atoms}}^g$  includes  $n_{\text{atoms}}$  embeddings and  $E_{\text{grids}}^g$  includes  $n_{\text{grids}}$  embeddings. For each position of the traps  $\mathbf{v}$ , we define learnable embeddings  $\mathbf{e}_g(\mathbf{v})$ . Then the element of  $E_{\text{atoms}}$  is

$$[E_{\text{atoms}}^g]_q = \begin{cases} \mathbf{e}_g(\mathbf{v}'_q) - \mathbf{e}_g(\mathbf{v}'_{q'}) & \text{the non-PA gates include } q \\ \mathbf{0} & \text{otherwise} \end{cases}, \quad (16)$$

where  $q'$  is the ID of the atom that couples with  $q$ -th atom in the non-PA gate. Let  $\mathbf{v}^{(1)} \dots \mathbf{v}^{(n_{\text{grids}})}$  be the positions of all traps. Then the element of  $E_{\text{grids}}$  is

$$[E_{\text{grids}}^g]_j = \mathbf{e}_g(\mathbf{v}^{(j)}) - \mathbf{e}_g(\mathbf{v}'_{q_0}). \quad (17)$$

The Gates Transformer is composed of multiple layers of (i) the multi-head attention, (ii) the layer norm, and (iii) the multi-layer perceptron. In the multi-head attention layer, the information of each vector at each position propagates to the output of different positions; how the information propagates is governed by the so-called attention mechanism [51], but the attention mask prohibits some propagation. We accept all the information flow at the same position, namely, the information at a position can be included in the output at that position. For the information flow to a different position, we only accept the flow from  $E_{\text{atoms}}^g$  to  $E_{\text{grids}}^g$ . In addition, we prohibit any flow from the position where  $[E_{\text{atoms}}^g]_q = \mathbf{0}$ . The attention mask is setup to realize the information flow and is used in all the multi-head attention layers.

The output of the Gates Transformer has the same structure as  $E_{\text{atoms}}^g \oplus E_{\text{grids}}^g$ , and we use the last  $n_{\text{grids}}$  vectors in the later calculation. We write the list of vectors obtained by applying Gates Transformer to the  $k$ -th input chunk as  $F^{(k)} := \{\mathbf{f}_j^{(k)}\}_{j=1}^{n_{\text{grids}}}$ .

*b. Moves transformer* The moves already planned are processed by the Moves Transformer. In this component, the Transformer layer captures correlations between already planned moves and potential moves of the current playable atom. The input to the Transformer is built via the moves embedder, producing  $E_{\text{atoms}}^m \oplus E_{\text{grids}}^m$ , where  $E_{\text{atoms}}^m$  includes  $n_{\text{atoms}}$  embeddings and  $E_{\text{grids}}^m$  includes  $n_{\text{grids}}$  embeddings. As in the case of the gates embedder, we define learnable embeddings  $\mathbf{e}_m(\mathbf{v})$  for each grid position  $\mathbf{e}_m$ . Then, the element of  $E_{\text{atoms}}^m$  is

$$[E_{\text{atoms}}^m]_q = \mathbf{e}_m(\mathbf{v}'_q) - \mathbf{e}_m(\mathbf{v}_q) \quad (18)$$

For  $E_{\text{grids}}^m$ , we set  $[E_{\text{grids}}^m]_j = \mathbf{e}_m(\mathbf{v}^{(j)}) - \mathbf{e}_m(\mathbf{v}_{q_0})$ .

The architecture of the Moves Transformer is the same as the Gates Transformer. The attention mask is also designed similarly. We accept all the information flow at the same position. For the information flow to a different position, we only accept the flow from  $E_{\text{atoms}}^m$  to  $E_{\text{grids}}^m$ . In addition, we prohibit any flow from the position where  $[E_{\text{atoms}}^m]_q = \mathbf{0}$ .

The output of the Moves Transformer has the same structure as  $E_{\text{atoms}}^m \oplus E_{\text{grids}}^m$ , and we use the last  $n_{\text{grids}}$  vectors in the later calculation. We write the list of vectors obtained by applying Moves Transformer as  $F^{(0)} := \{\mathbf{f}_j^{(0)}\}_{j=1}^{n_{\text{grids}}}$ .

*c. Readout* The outputs from Gates Transformer and Moves transformer is aggregated by the following mean pooling step to  $F = \{\mathbf{f}_j\}_{j=1}^{n_{\text{grids}}}$ , where

$$\mathbf{f}_j := \frac{1}{K+1} \sum_{k=0}^K \mathbf{f}_j^{(k)}. \quad (19)$$

## 3. The QC-Daemon's policy and its value function

$F$  and  $\mathbf{d}$  are used to calculate the policy (QC-Daemon). We also describe how the value function is constructed for the actor-critic type RL algorithm.

*a. Policy (QC-Daemon)* For each position, MLP is applied to  $\mathbf{f}_j$  for each  $j$ , and converted to one-dimensional output  $o_j$ . The logit is defined by  $w_j := o_j + d_j$ . The grid ID  $j$  is sampled according to the probability proportional to  $\exp(w_j)$ , but the grid IDs filled with other atom are masked.

*b. Value* We first mask the grid IDs filled with other atom; let  $\{j_1, \dots, j_m\}$  be the list of IDs unmasked, then one vector  $\mathbf{f}$ , which is the average of  $\mathbf{f}_{j_1} \dots \mathbf{f}_{j_m}$  is calculated. Then, MLP with one-dimensional output is applied to  $\mathbf{f}$  and another MLP with one-dimensional output is applied to  $\mathbf{d}$ . Finally, these outputs are summarized as the output of the value function.

## V. NUMERICAL EXPERIMENTS

In this section, we show two types of numerical experiments. In the first experiment in Section V A, we train the model with one circuit and apply it to that same circuit. In the second experiment in Section V B, we first train the model with 30 circuits with different numbers of qubits and apply it to new unseen problems to verify the transferability of our model.

### A. Training from scratch

We use quantum circuits for quantum Fourier transform (qft), quantum neural network (qnn), variational quantum eigensolver (groundstate), quantum approximate optimization algorithm (qaoa) from the MQT Bench [52] and generate Hamiltonian evolution circuits for the Fermi-Hubbard model [53] (hubbard). For the Fermi-Hubbard model simulation, we use the first-order Lie-Trotter formula to decompose the time evolution, repeating the operation twice. Specifically, for a given time evolution  $e^{iHt}$  with  $H$  as the Hamiltonian having the Pauli decomposition  $H = \sum_j c_j P_j$ , we obtain the circuit for our simulation by

$$\left( \prod_j e^{ic_j P_j \Delta t} \right)^{t/\Delta t}, \quad c_j \in \mathbb{R} \text{ and } t/\Delta t = 2.$$

For calculating the reconfiguration cost of  $L$  and  $G$ , we set the physical parameters in Eq. (8) to  $\alpha = 0.02$  and  $\beta = 0.002$ . We use proximal policy optimization (PPO) [54] as the RL algorithm. The other hyper-parameters are listed in Appendix A. The training is performed on a system equipped with eight NVIDIA H100 Tensor Core GPU.

In Fig. 5, we show the cost reduction ratio at each iteration for each benchmark problem. The horizontal axis represents the number of training iterations, and the vertical axis indicates the cost reduction compared to the case where no reconfiguration is performed, which corresponds to the gray dotted line. Each green line corresponds to the training with a different seed for the random generation of the initial configuration and model parameters. We see that the cost reduction is negative at the beginning of training, which means that the generated actions increase the cost due to unnecessary layout changes. However, after some iterations, the cost reduction becomes positive, and we can achieve a reduction in cost of up to about 20% through layout changes. In Fig. 6, we show the number of actions that change the layout of atoms; namely, we count only the actions that specify a position for a playable atom different from its current grid. We see that at the beginning of training, many actions are generated, but only a few remain after training. This implies that the QC-Daemon successfully learns how to generate effective layout changes.

### B. Transferability evaluation

We also conducted an experiment to evaluate the transferability of QC-Daemon. In this experiment, the model was built using only dynamic feature extraction, allowing it to generalize across different qubit counts and circuit structures. An attention mechanism, which enables the model to handle variable-length inputs, makes this possible.

As the training dataset, we generated 30 Hamiltonians, each consisting of 40 terms of local random Pauli operators acting on 40–50 qubits. These were converted into quantum circuits by applying 4 steps of Trotterization, and used for training. During training, one Hamiltonian was randomly selected from the 30, and the model was executed starting from a random initial configuration. The model was then updated based on the execution result, using the same hyperparameters as in previous experiments.

As the test dataset, we additionally generated three new test circuits not included in the training dataset with 40–50 qubits using the same procedure as described above. We compared the best outcomes from 1-shot, 10-shot, and 100-shot evaluations to those obtained without training, where  $n$ -shot denotes the best result obtained from  $n$  independent runs of the trained model on the same circuit.

We also repeated the same procedure using larger Hamiltonians with 60 terms acting on 80–100 qubits, and studied the performance. The training is performed on a system with six NVIDIA H100 Tensor Core GPUs.

In Fig. 7, we show the change in the mean reward for each iteration during the training of the transferable models with 40–50 qubit circuit datasets (left) and 80–100 qubit circuit datasets (right). The reward is calculated as the cost reduction compared to the case without reconfiguration. A positive reward indicates that the cost is reduced compared to the case without reconfiguration. The mean reward is calculated as the moving average over the past 50 steps. In both cases, the mean reward becomes positive after several thousand iterations, indicating that the model successfully learns generally effective reconfiguration strategies across multiple circuits.

Table I and Table II show the cost reduction for test circuits with and without transfer for the 40–50 qubit circuit datasets (left) and the 80–100 qubit circuit datasets, for each number of shots. Even in the 1-shot case, we observe that the model trained with the datasets can successfully generate practical reconfiguration steps for unseen circuits, though the untrained model generates unreasonable layout changes. Table III shows the duration (seconds) for generating all the actions for each number of shots using the same test circuit as in Table II on a system equipped with an NVIDIA H100 Tensor Core GPU. We also show the number of chunks in the same table. We see that the difference in execution time between 1 shot and 10 shots is smaller than a factor of two due to efficient parallel computation. The execution time

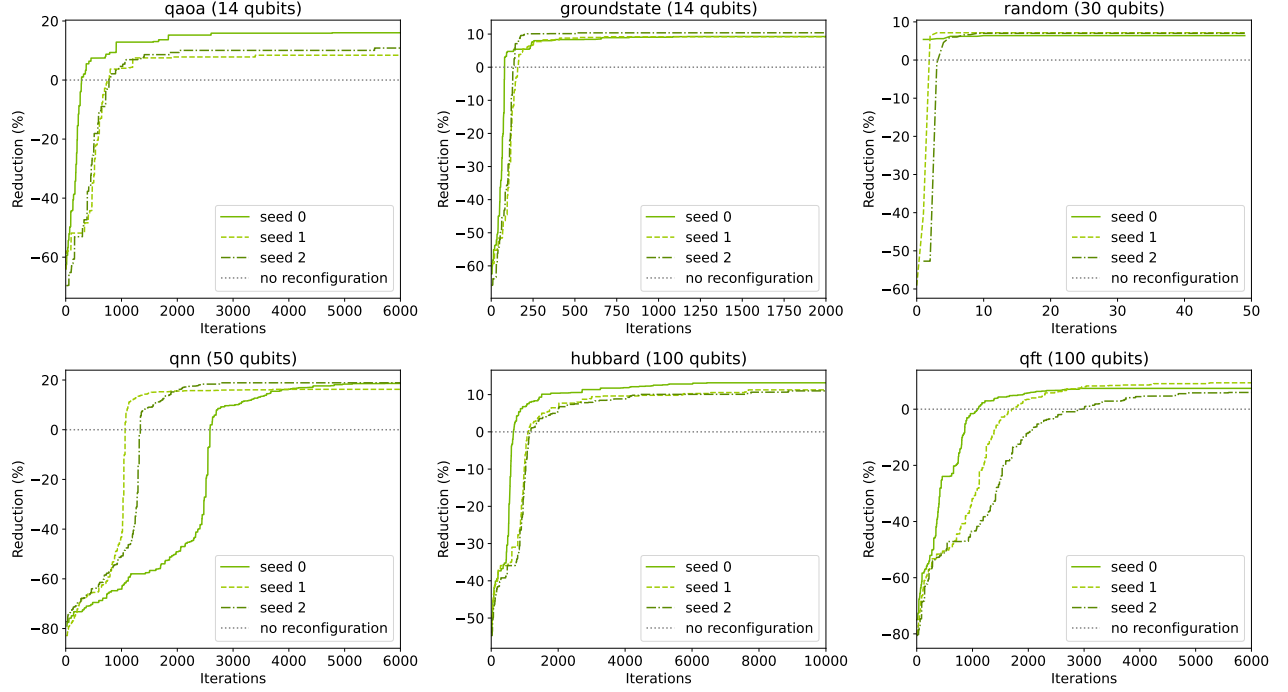


FIG. 5. Cost reduction (%) for each benchmark across training iterations, as achieved by our QC-Daemon in the experiment described in Section V A. The model is trained from scratch and applied to the same circuit. The horizontal axis shows the number of training iterations, while the vertical axis indicates the cost reduction relative to the baseline without reconfiguration (gray dotted line). Each green line represents a training run with a different random seed for initializing the configuration and model parameters.

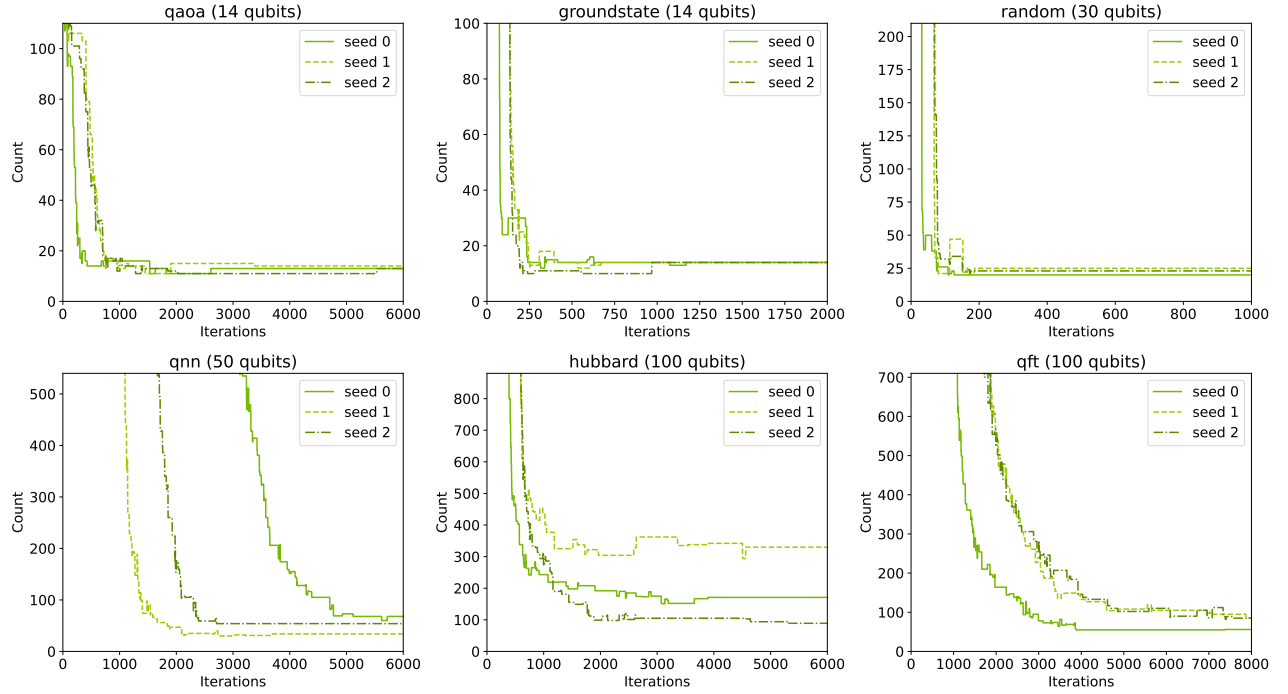


FIG. 6. The number of reconfiguration actions for each benchmark at each iteration in the same experiment as Fig. 5. We count only the actions that result in a playable atom being placed at a different grid location, thereby changing the overall layout.

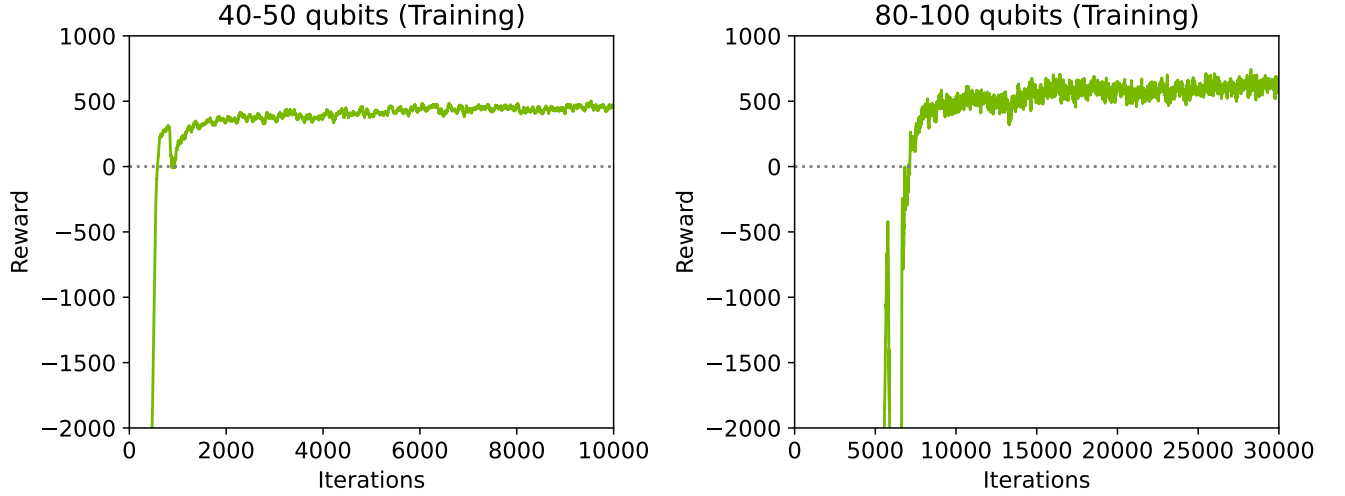


FIG. 7. The change in the mean reward for each iteration during the training of the transferable models with 40–50 qubit circuit datasets (left) and 80–100 qubit circuit datasets (right). The mean reward is calculated as the moving average over the past 50 steps. A positive reward indicates that the cost is reduced compared to the case without reconfiguration.

with 100 shots scales slightly worse from the 10-shot case, but is still better than linear scaling. We note that most of the computational time was dedicated to the action generation from the model on the GPU. If we use a reward that requires intensive CPU resources, the scaling behaviour would degrade.

TABLE I. Cost reduction with and without transfer in the 40–50 qubit experiments for each number of shots.

Seed Qubits	Transfer	1 shot	10 shots	100 shots
0	No	-74.95%	-72.91%	-72.31%
	Yes	8.36%	8.36%	8.90%
1	No	-78.42%	-76.77%	-73.85%
	Yes	7.42%	8.52%	10.33%
2	No	-62.85%	-62.81%	-57.49%
	Yes	8.32%	8.60%	9.20%

TABLE II. Results of the same experiment as in Table I, but conducted with 80–100 qubit circuits for each number of shots.

Seed Qubits	Transfer	1 shot	10 shots	100 shots
0	No	-86.98%	-86.43%	-85.31%
	Yes	6.66%	8.51%	8.51%
1	No	-78.37%	-78.37%	-77.21%
	Yes	7.90%	8.66%	8.85%
2	No	-92.18%	-91.64%	-90.86%
	Yes	7.06%	7.78%	8.57%

The transferability shown in the experiment is practically essential, as it provides a way to generate moves without additional training. While we restrict the

TABLE III. The duration (seconds) for generating all the actions for each number of shots using the same test circuit as in Table II on a system equipped with an NVIDIA H100 Tensor Core GPU.

Seed Qubits	Chunks	1 shot	10 shots	100 shots	
0	97	436	12.83	18.32	87.02
1	84	490	12.45	17.44	81.74
2	92	335	13.12	17.60	85.84

dataset class in this demonstration, it would be beneficial to pre-train the model on various quantum circuits as a foundation model for generating layout changes in future research.

## VI. DISCUSSIONS AND CONCLUSIONS

In this paper, we introduced the QC-Game as a framework based on Markov decision processes to dynamically control the state of a device while executing a given quantum circuit. We then propose QC-Daemon as an agent to solve the QC-Game. An essential application of the QC-Game is the Atom Game, which involves planning the sequence of reconfiguring the layout of atoms in reconfigurable neutral atom arrays. The actual physical operations for reconfiguration and gate execution in the Atom Game are handled abstractly, making the approach compatible with any reconfiguration or gate-execution protocol. We demonstrated that our reinforcement-learning agent, QC-Daemon, constructed with two types of transformers, can reduce the logarithmic infidelity compared to the scenario where no layout

change is made, across various benchmark problems. We also demonstrate the transferability of our model using datasets of 40–50 qubits and 80–100 qubits. This transferability is beneficial for rapidly generating physical operations in practical applications.

There are multiple directions for future research. As we note in Section I, this work focuses on optimizing the reconfiguration step to simplify the problem structure. However, other stages in the compilation pipeline may become more important depending on the nature of the target circuit such as initialization. Extending our AI compiler to support additional stages is an important direction for future research.

Our reward provides a rough estimate of the infidelity, and we do not perform a full simulation of scheduling gates or reconfigurations on an actual quantum device. Performing such simulations would require assuming specific details of a real scheduler. In principle, this would only involve modifying the definitions of  $G(s_t, C_t)$  and  $L(s_t, s_{t+1})$ . Additionally, in our setting, the reconfiguration/gate execution protocols are given as inputs. However, another possibility is training the agent to produce the physical atom moves for reconfiguration or gate execution directly, given a particular atom layout. The application for fault-tolerant quantum computation (FTQC) is also an interesting direction. We expect that the same argument applies in FTQC by using a logical qubit instead of a physical one as the unit

of the atom move; however, additional operations necessary for a logical processor, such as the implementation of non-transversal gates, will affect the advantage of our QC-Daemon.

## ACKNOWLEDGEMENT

A related work [37] considering the reconfiguration process was posted almost simultaneously with this manuscript. Although developed independently, their study addresses a similar problem and provides complementary insights. We acknowledge their timely contribution to this rapidly evolving field.

The authors thank early discussions with Shengtao Wang and Casey Duckering. H.H. acknowledge support from the NSERC-Google Industrial Research Chair award. L.M.C. acknowledges support from the Novo Nordisk Foundation, NNF Quantum Computing Programme. K.P. acknowledges the generous support of the Canada 150 Research Chairs Program through A.A.-G. A.A.-G. thanks Anders G. Frøseth for his generous support and acknowledges the generous support of Natural Resources Canada and the Canada 150 Research Chairs program. This research is part of the University of Toronto’s Acceleration Consortium, which receives funding from the CFREF-2022-00042 Canada First Research Excellence Fund.

- 
- [1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.
  - [2] G. C. Pflug and A. Pichler, *Multistage stochastic optimization*, vol. 1104. Springer, 2014.
  - [3] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson, *et al.*, “Pddl—the planning domain definition language,” *Technical Report, Tech. Rep.*, 1998.
  - [4] D. McDermott, “The formal semantics of processes in pddl,” in *Proc. ICAPS Workshop on PDDL*, pp. 101–155, Citeseer, 2003.
  - [5] R. S. Sutton, “Reinforcement learning: An introduction,” *A Bradford Book*, 2018.
  - [6] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, “An introduction to deep reinforcement learning,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.
  - [7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
  - [8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
  - [9] H. Levine, A. Keesling, G. Semeghini, A. Omran, T. T. Wang, S. Ebadi, H. Bernien, M. Greiner, V. Vuletić, H. Pichler, and M. D. Lukin, “Parallel implementation of high-fidelity multiqubit gates with neutral atoms,” *Phys. Rev. Lett.*, vol. 123, p. 170503, Oct 2019.
  - [10] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, *et al.*, “Logical quantum processor based on reconfigurable atom arrays,” *Nature*, vol. 626, no. 7997, pp. 58–65, 2024.
  - [11] S. J. Evered, D. Bluvstein, M. Kalinowski, S. Ebadi, T. Manovitz, H. Zhou, S. H. Li, A. A. Geim, T. T. Wang, N. Maskara, *et al.*, “High-fidelity parallel entangling gates on a neutral-atom quantum computer,” *Nature*, vol. 622, no. 7982, pp. 268–272, 2023.
  - [12] G. A. team, “Quantum error correction below the surface code threshold,” *Nature*, vol. 638, p. 920–926, Dec. 2024.
  - [13] N. Berthussen, J. Dreiling, C. Foltz, J. P. Gaebler, T. M. Gatterman, D. Gresh, N. Hewitt, M. Mills, S. A. Moses, B. Neyenhuis, P. Siegfried, and D. Hayes, “Experiments with the four-dimensional surface code on a quantum charge-coupled device quantum computer,” *Physical Review A*, vol. 110, Dec. 2024.
  - [14] N. Lacroix, A. Bourassa, F. J. Heras, L. M. Zhang, J. Bausch, A. W. Senior, T. Edlich, N. Shuttly, V. Sivak, A. Bengtsson, *et al.*, “Scaling and logic in the color code on a superconducting quantum processor,” *arXiv preprint arXiv:2412.14256*, 2024.
  - [15] H. Aghaee Rad, T. Ainsworth, R. Alexander, B. Altieri, M. Askarani, R. Baby, L. Banchi, B. Baragiola, J. Bourassa, R. Chadwick, *et al.*, “Scaling and network-



- ing a modular photonic quantum computer,” *Nature*, pp. 1–8, 2025.
- [16] B. Tan and J. Cong, “Optimality study of existing quantum computing layout synthesis tools,” *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1363–1373, 2021.
  - [17] M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins, “Using reinforcement learning to perform qubit routing in quantum compilers,” *ACM Transactions on Quantum Computing*, vol. 3, p. 1–25, May 2022.
  - [18] Y. Alexeev, M. H. Farag, T. L. Patti, M. E. Wolf, N. Ares, A. Aspuru-Guzik, S. C. Benjamin, Z. Cai, Z. Chandani, F. Fedele, *et al.*, “Artificial intelligence for quantum computing,” *arXiv preprint arXiv:2411.09131*, 2024.
  - [19] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, “Optimized compilation of aggregated instructions for realistic quantum computers,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS ’19, (New York, NY, USA), p. 1031–1044, Association for Computing Machinery, 2019.
  - [20] D. B. Tan, W.-H. Lin, and J. Cong, “Compilation for dynamically field-programmable qubit arrays with efficient and provably near-optimal scheduling,” *arXiv preprint arXiv:2405.15095*, 2024.
  - [21] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, “Elementary gates for quantum computation,” *Physical review A*, vol. 52, no. 5, p. 3457, 1995.
  - [22] A. W. Harrow, B. Recht, and I. L. Chuang, “Efficient discrete approximations of quantum gates,” *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4445–4451, 2002.
  - [23] C. M. Dawson and M. A. Nielsen, “The solovay-kitaev algorithm,” *arXiv preprint quant-ph/0505030*, 2005.
  - [24] L. Moro, M. G. Paris, M. Restelli, and E. Prati, “Quantum compiling by deep reinforcement learning,” *Communications Physics*, vol. 4, no. 1, p. 178, 2021.
  - [25] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, “Quantum circuit optimization with deep reinforcement learning,” *arXiv preprint arXiv:2103.07585*, 2021.
  - [26] D. Kremer, V. Villar, H. Paik, I. Duran, I. Faro, and J. Cruz-Benito, “Practical and efficient quantum circuit synthesis and transpiling with reinforcement learning,” *arXiv preprint arXiv:2405.13196*, 2024.
  - [27] B. Tan and J. Cong, “Optimality study of existing quantum computing layout synthesis tools,” *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1363–1373, 2020.
  - [28] D. Bluvstein, H. Levine, G. Semeghini, T. T. Wang, S. Ebadi, M. Kalinowski, A. Keesling, N. Maskara, H. Pichler, M. Greiner, *et al.*, “A quantum processor based on coherent transport of entangled atom arrays,” *Nature*, vol. 604, no. 7906, pp. 451–456, 2022.
  - [29] H. Levine, D. Bluvstein, A. Keesling, T. T. Wang, S. Ebadi, G. Semeghini, A. Omran, M. Greiner, V. Vuletić, and M. D. Lukin, “Dispersive optical systems for scalable raman driving of hyperfine qubits,” *Phys. Rev. A*, vol. 105, p. 032618, Mar 2022.
  - [30] L. M. K. Vandersypen and I. L. Chuang, “Nmr techniques for quantum control and computation,” *Rev. Mod. Phys.*, vol. 76, pp. 1037–1069, Jan 2005.
  - [31] B. Tan and J. Cong, “Optimal layout synthesis for quantum computing,” in *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.
  - [32] B. Tan, D. Bluvstein, M. D. Lukin, and J. Cong, “Qubit mapping for reconfigurable atom arrays,” in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD ’22, (New York, NY, USA), Association for Computing Machinery, 2022.
  - [33] J. Ruan, X. Fang, H. Zhang, A. Li, T. Humble, and Y. Ding, “Powermove: Optimizing compilation for neutral atom quantum computers with zoned architecture,” *arXiv preprint arXiv:2411.12263*, 2024.
  - [34] J. Ludmir and T. Patel, “Parallax: A compiler for neutral atom quantum computers under hardware constraints,” in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–17, IEEE, 2024.
  - [35] W.-H. Lin, D. B. Tan, and J. Cong, “Reuse-aware compilation for zoned quantum architectures based on neutral atoms,” in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 127–142, IEEE, 2025.
  - [36] Y. Stade, L. Schmid, L. Burgholzer, and R. Wille, “An abstract model and efficient routing for logical entangling gates on zoned neutral atom architectures,” in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, p. 784–795, IEEE, Sept. 2024.
  - [37] Y. Stade, W.-H. Lin, J. Cong, and R. Wille, “Routing-aware placement for zoned neutral atom-based quantum computing,” *arXiv preprint arXiv:2505.22715*, 2025.
  - [38] Q. Xu, J. P. Bonilla Ataides, C. A. Pattison, N. Raveendran, D. Bluvstein, J. Wurtz, B. Vasić, M. D. Lukin, L. Jiang, and H. Zhou, “Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays,” *Nature Physics*, vol. 20, p. 1084–1090, Apr. 2024.
  - [39] [https://github.com/QuEraComputing/reconfiguration\\_cost\\_estimator](https://github.com/QuEraComputing/reconfiguration_cost_estimator).
  - [40] R. L. Brooks, “On colouring the nodes of a network,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 37, p. 194–197, Apr. 1941.
  - [41] P. S. Rodriguez, J. M. Robinson, P. N. Jepsen, Z. He, C. Duckering, C. Zhao, K.-H. Wu, J. Campo, K. Bagnall, M. Kwon, *et al.*, “Experimental demonstration of logical magic state distillation,” *arXiv preprint arXiv:2412.15165*, 2024.
  - [42] A. Y. Kitaev, “Fault-tolerant quantum computation by anyons,” *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.
  - [43] C. Nayak, S. H. Simon, A. Stern, M. Freedman, and S. Das Sarma, “Non-abelian anyons and topological quantum computation,” *Reviews of Modern Physics*, vol. 80, no. 3, pp. 1083–1159, 2008.
  - [44] R. Raussendorf and J. Harrington, “Fault-tolerant quantum computation with high threshold in two dimensions,” *Physical review letters*, vol. 98, no. 19, p. 190504, 2007.
  - [45] D. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, “Surface code quantum computing by lattice surgery,” *New Journal of Physics*, vol. 14, no. 12, p. 123011, 2012.
  - [46] D. Litinski, “A game of surface codes: Large-scale quantum computing with lattice surgery,” *Quantum*, vol. 3, p. 128, 2019.
  - [47] A. Paler and A. G. Fowler, “Opensurgery for topological assemblies,” 2020.
  - [48] M. Beverland, V. Kliuchnikov, and E. Schoute, “Surface code compilation via edge-disjoint paths,” *PRX Quantum*, vol. 3, no. 2, p. 020342, 2022.

- [49] G. Watkins, H. M. Nguyen, K. Watkins, S. Pearce, H.-K. Lau, and A. Paler, “A high performance compiler for very large scale surface code computations,” *Quantum*, vol. 8, p. 1354, 2024.
- [50] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, *et al.*, “Mlp-mixer: An all-mlp architecture for vision,” *Advances in neural information processing systems*, vol. 34, pp. 24261–24272, 2021.
- [51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [52] N. Quetschlich, L. Burgholzer, and R. Wille, “MQT Bench: Benchmarking software and design automation tools for quantum computing,” *Quantum*, 2023. MQT Bench is available at <https://www.cda.cit.tum.de/mqtbench/>.
- [53] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, A. Bengtsson, S. Boixo, M. Broughton, B. B. Buckley, *et al.*, “Observation of separated dynamics of charge and spin in the fermi-hubbard model,” *arXiv preprint arXiv:2010.07965*, 2020.
- [54] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

## Appendix A: Hyper-parameter of the numerical experiment

### 1. Machine learning model parameters

TABLE IV. Model architecture, training, and experimental parameters. The device parameters are used to compute a dimensionless quantity, the infidelity. Therefore, while each parameter may originally have a physical unit, the units are omitted in the description, and only scalar values are shown.

Category	Parameters
<b>Dynamic feature:</b> Gate transformer	Embedding dimension = 20 Number of layers = 3 Number of heads in multi-head attention = 4
<b>Dynamic feature:</b> Move transformer	Embedding dimension = 20 Number of layers = 3 Number of heads in multi-head attention = 4
<b>Static feature</b>	Player embedding dimension = 40 Time embedding dimension = 40 Board embedding dimension = 10 Number of layers in MLP-Mixer = 1
<b>Training parameters:</b> PPO	Clipping coefficient ( $\epsilon$ ) = 0.2 Entropy coefficient ( $c_{\text{entropy}}$ ) = 0.01 Value function coefficient ( $c_{\text{vf}}$ ) = 0.5 Maximum gradient norm = 0.5 Learning rate = 0.00025
<b>Device parameters:</b>	Acceleration constant ( $\gamma$ ) = 1 Inverse coherence time ( $\alpha$ ) = 0.02 Atom loss parameter ( $\beta$ ) = 0.02 Inter-zone transfer time ( $T_G$ ) = 10 Lattice spacing of the grid = 1
<b>Other parameters</b>	Window size ( $W$ ) = 2 Horizon length ( $K$ ) = 5

## 2. Size of the grids

TABLE V. The list of the size of the grids in each benchmark experiment. The benchmark 'transfer' represents the experiment in Section [VB](#).

Benchmark	Qubits	Grid size (row $\times$ col)
qaoa	14	$4 \times 10$
groundstate	14	$4 \times 10$
random	30	$7 \times 10$
qnn	50	$5 \times 20$
qft	100	$10 \times 20$
hubbard	100	$10 \times 20$
transfer-learning	40-50	$5 \times 20$
transfer-learning	80-100	$10 \times 20$